PSEE Architecture Report

Attachment - Section 11

Architectures and Models

for Next Generation Process-based

Software Engineering Environments

TRW Systems Integration Group
Redondo Beach, California 90278

Sponsored by

Advanced Research Projects Agency (ARPA)
and Space and Naval Warfare Systems Command
ARPA Order No. B343
Under SPAWAR Contract # N00039-95-C-0017

February 1995

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)

2. REPORT DATE
February 1995

3. REPORT TYPE AND DATES COVERED
Interim

4. TITLE AND SUBTITLE
PSEE Architecture Report
PSEE Architecture Report Attachment - Section 11

5. FUNDING NUMBERS
SPAWAR C#
N00039-95-C-0017
ARPA Order #B343

6. AUTHOR(S)
Maria H. Penedo (author, editor)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
TRW
One Space Park
Redondo Beach, CA 90278

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
SPAWAR, 2451 Crystal Drive, Arlington, VA 22245-54200
ARPA, 3701 North Fairfax Drive, Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Statement A: Approved for public release, distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

This Architecture Report documents investigations towards the definition of a Process-based Software Engineering Environment (PSEE) Reference Architecture. Those investigations are in support of the definition of a component-based architectural approach for the rapid construction of PSEEs. It represents work in progress.

These investigations were conducted within the "Architectures and Models for Next Generation Process-based SEEs" project; they also revise and enhance prior related work. This document is to be the first in a series of Architecture Reports to be delivered by this contract (pending continuous funding).

This document actually consists of eleven (11) reports and four (4) references which provide technical data related to the current state (art and practice) of SEEs including architectural recommendations, requirements, and lessons learned; the data was gathered from national and international efforts and systems, and includes recent developments in the commercial sector and research programs. A key underlying assumption of this work is community participation and community consensus; therefore, some of the documents have been jointly developed with members of the community.

14. SUBJECT TERMS
architectures, process, software engineering environment

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

Trends in the Construction of Next Generation Software Engineering Environments

Maria Heloisa Penedo
TRW

Architecture Report Attachment (item 11)

February 1995

Outline

Introduction

- Rationale - Cost and Productivity Highlights
- Process - concepts, examples
- CASE/SEEs - concepts

SEEs: Models, Characterizations and Examples

- Concepts
- Reference Models
- SEE architectures types and Examples
- Integration
- Lessons Learned

Conclusions

Trends

Architectures:

- Client-Server
- Distribution
- Autonomy
- Interoperability
- Active
- Component-based
- Process-based
- User-tailorable

Processes:

- Architecture-driven
- Reused-based
- Design by Teams
- Cooperative (CSCW)

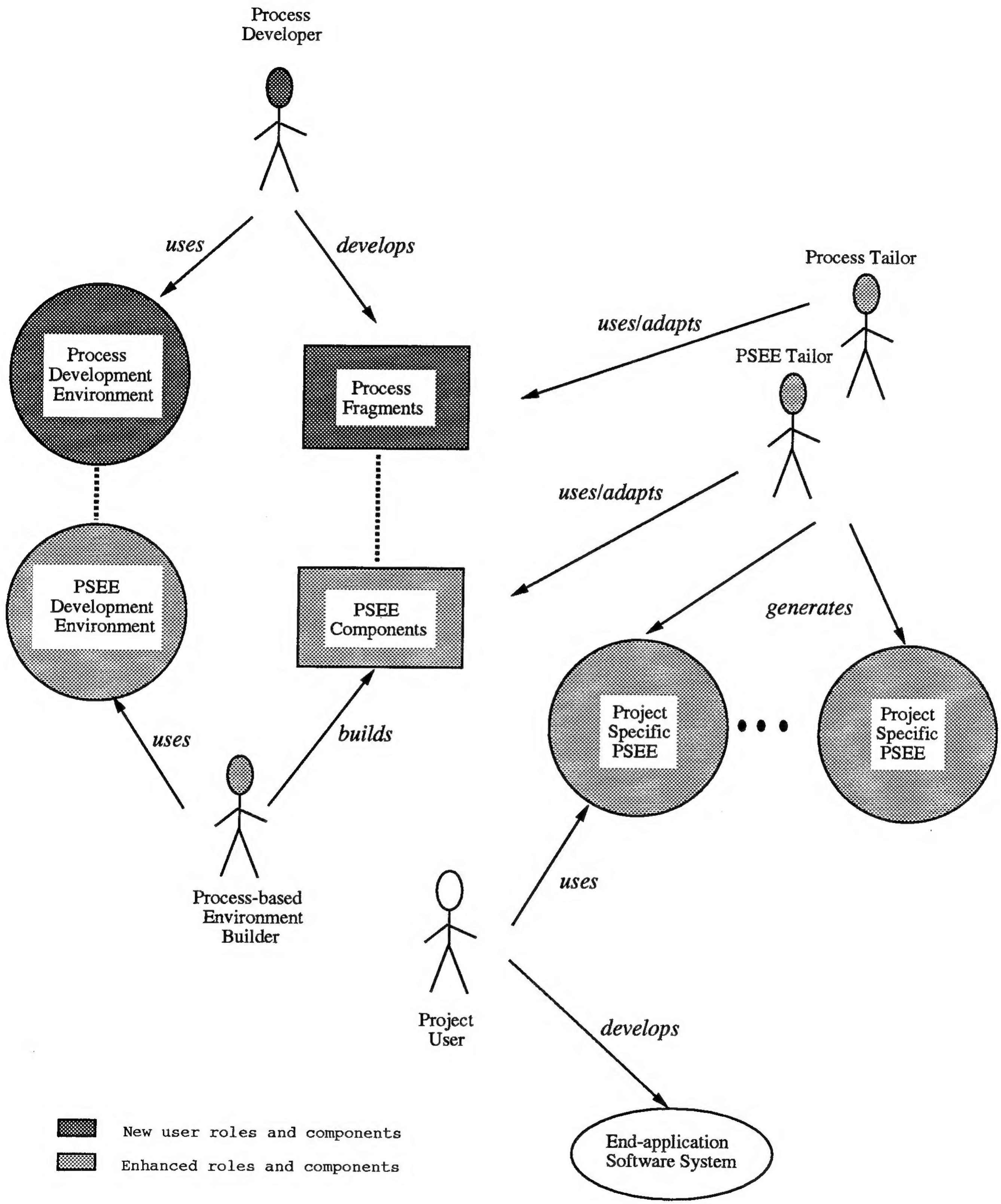
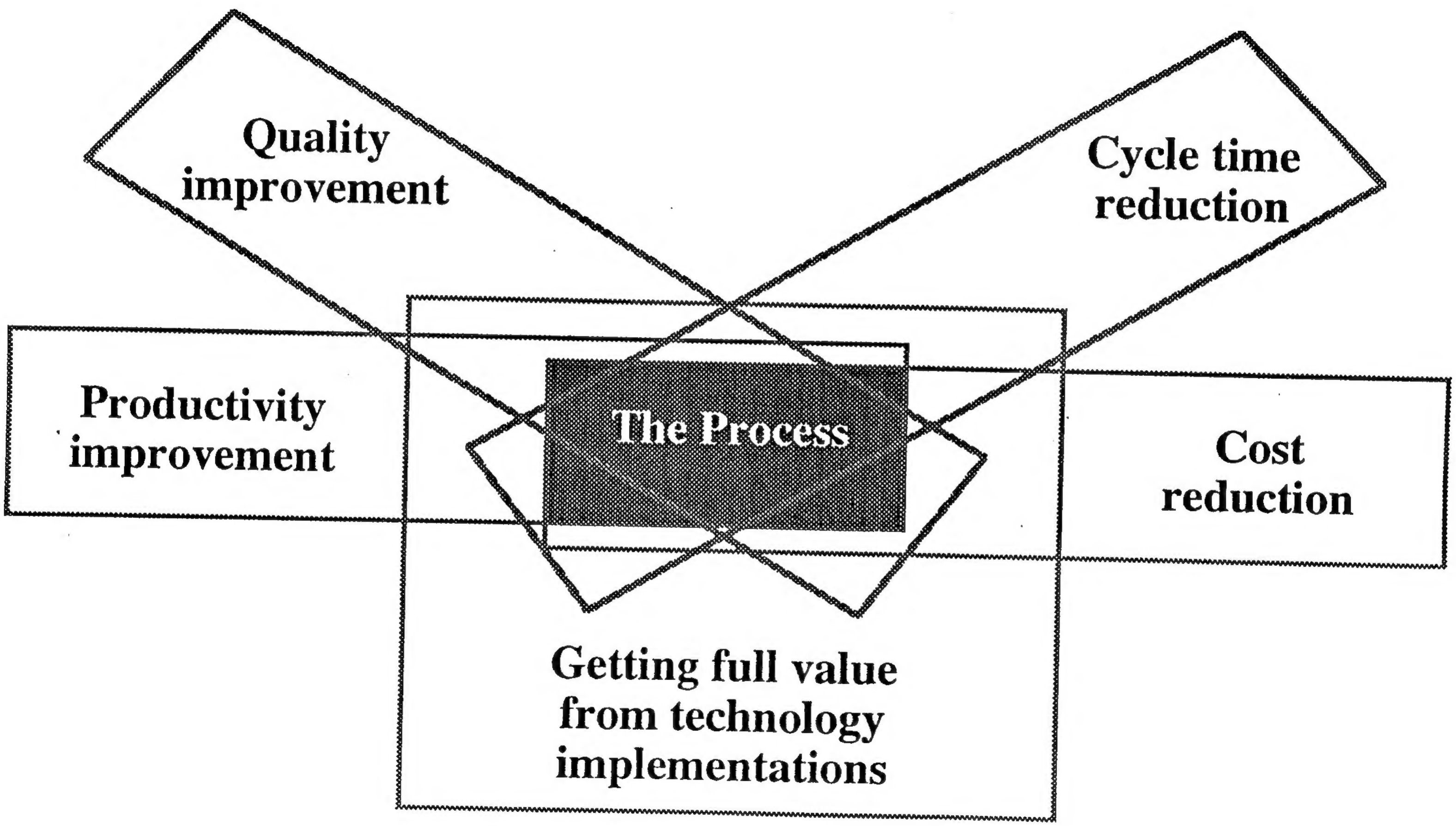


Figure 2: Adding process to SEEs: User Roles and Componentry

Focus on the Process

TRW



Outline

Introduction

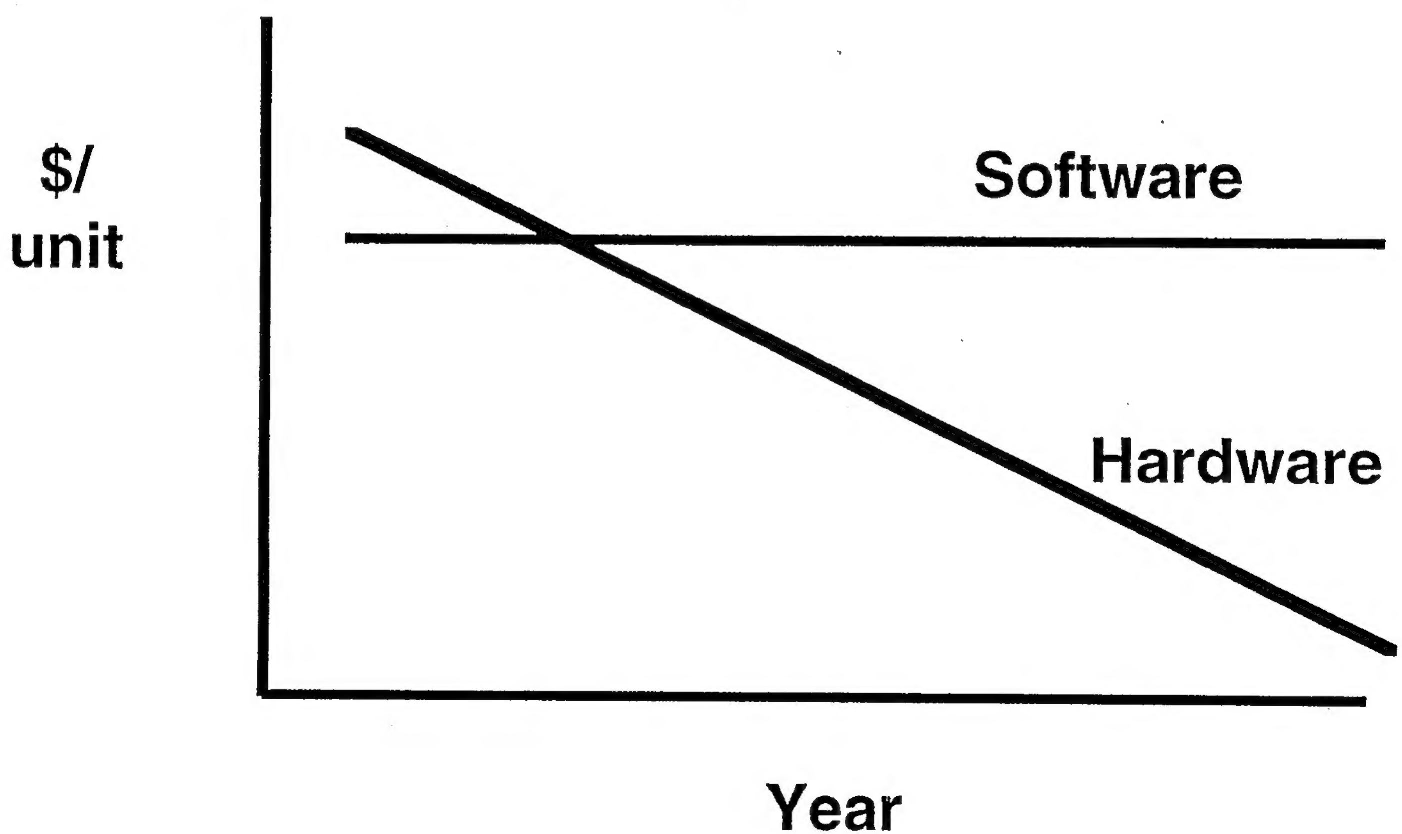
- Rationale - Cost and Productivity Highlights
- Process - concepts, examples
- CASE/SEEs - concepts

SEEs: Models, Characterizations and Examples

- Concepts
- Reference Models
- SEE architectures types and Examples
- Integration
- Lessons Learned

Conclusions

Typical Hardware-Software Cost Trend Comparison



Productivity - Concepts

- Webster: "Productive means furnishing results, benefits, or profits".
- Productivity: Which person (or team or organization) turns out the most: source statements, function points, or delivered systems.
- Problem: lack of a "good" unit of measure - none is universally accepted.

  Ex.: lines of code, cost per defect, percentages associated with phases.
- "Does CASE and/or COTS improve productivity?" Most say *yes*.
- Factors influencing software development productivity:
  - common, rigorous use of standardized development methods
  - better programming practices and languages (e.g., Ada, 4GL)
  - CASE tools, ...
- Cost Models provide powerful insights [Boehm, Jones]
- Keep in mind that motivation/objectives have a driving force (Weinberg-Schulman Experiment)

The impact of Tools [Jones]

- Application - Ada compiler
- To be developed by reasonably experienced programmers
- Best: many integrated tools; Worst: very few tools

	Best Environment	Average Environment	Worst Environment
Size in lines of C code	100,000	100,000	100,000
Total schedule	19.5	22	22.5
Person-months	790	1,367	1,953
Staff size	68	105	144
Cost per line	$40.45	$71.40	$103.30
Lines per month	126	73	51

Effect of Objectives on Productivity

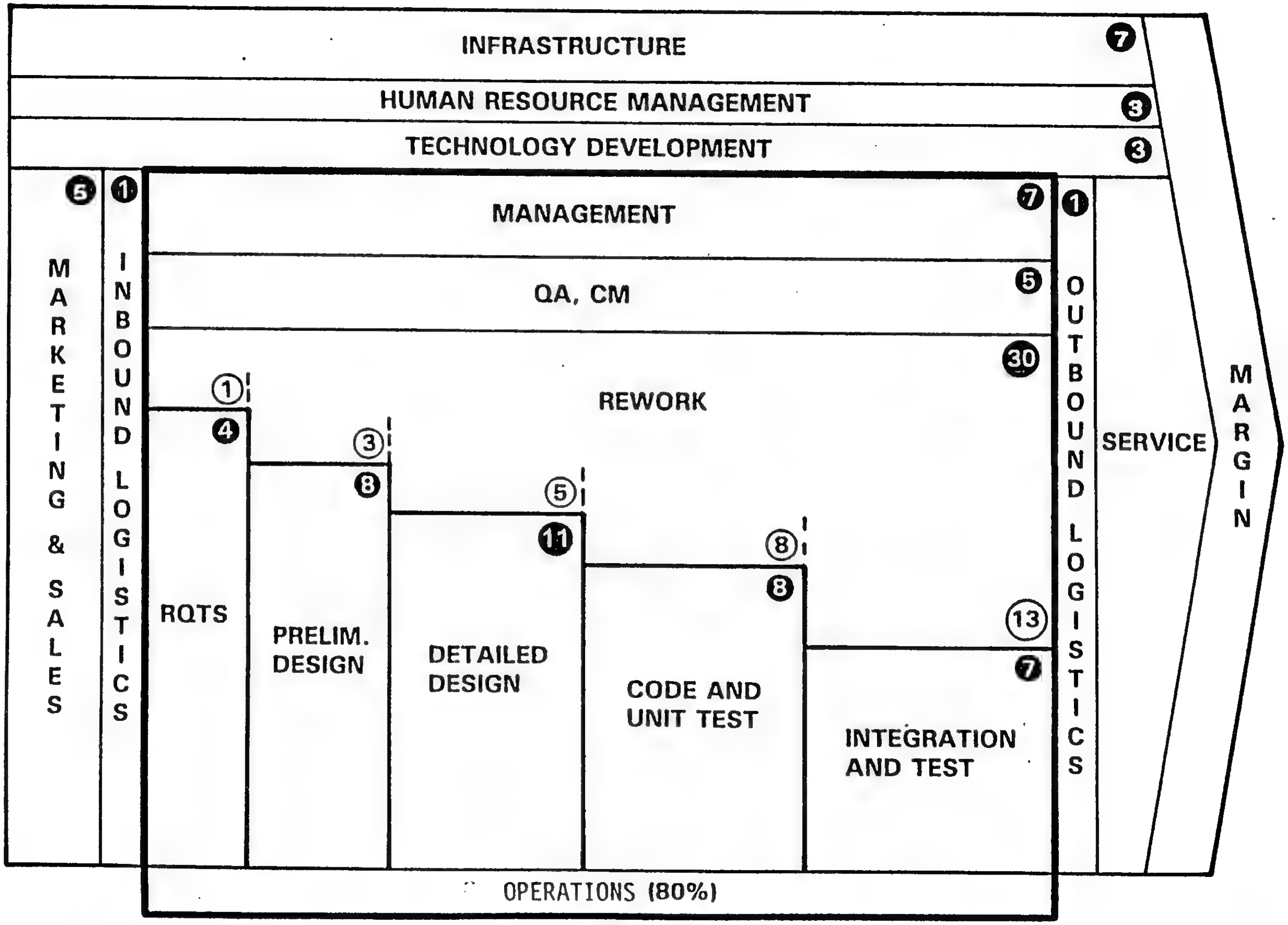
Weinberg-Schulman Experiment [1974]

Team Objective: Optimize	Number of Statements	Man-Hous	Productivity (State/M-H)
Core Memory	52	74	0.7
Number of Statements	33	30	1.1
Execution Time	100	50	2.0
Program Clarity	90	40	2.2
Programming Man-Hours	126	28	4.5
Output Clarity	166	30	5.5

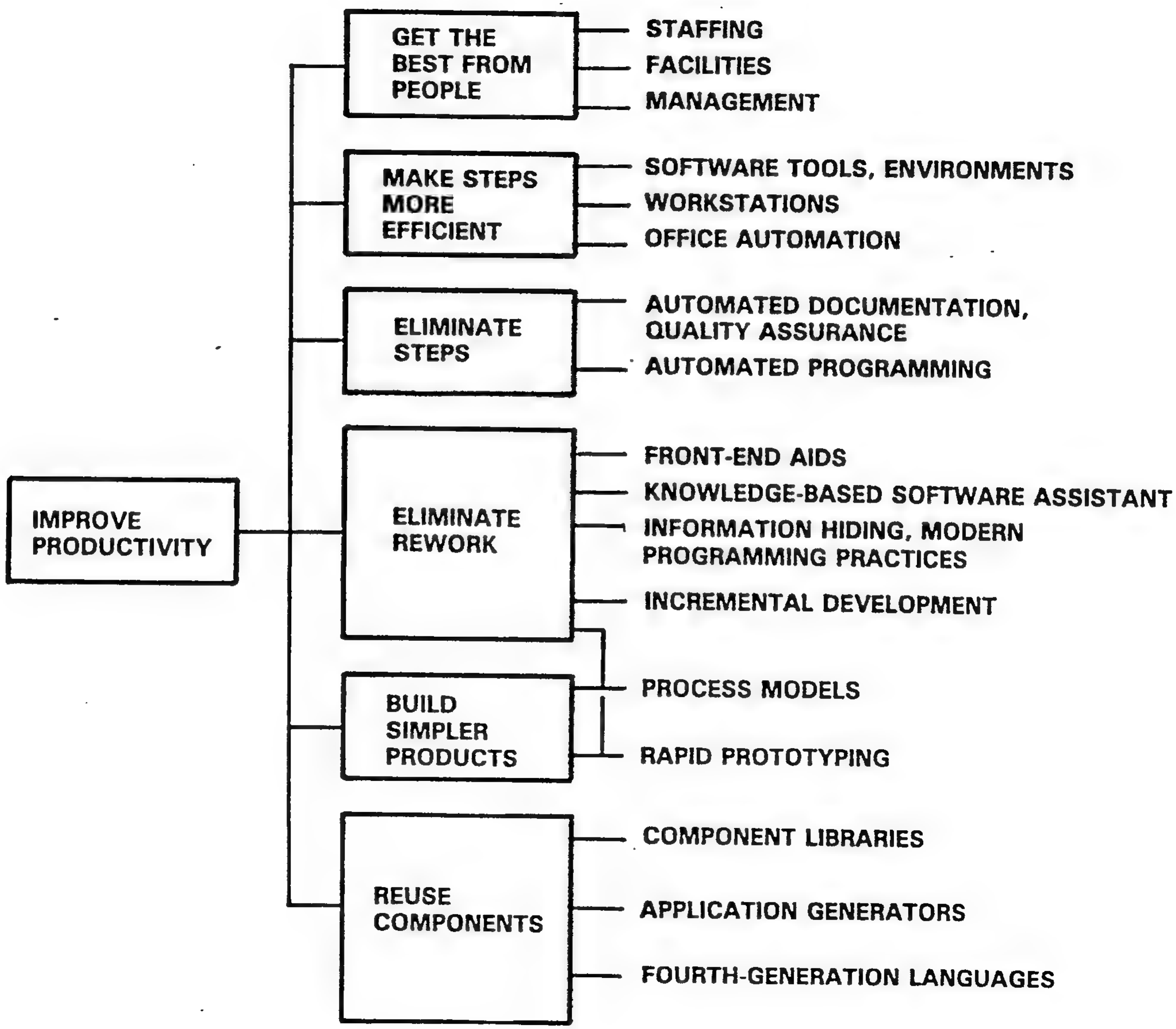
Value Chain Analysis

- based on Boehm's Improving Software Productivity paper.
- Value Chain Analysis was developed at Harvard Business School
  - useful method for understanding and controlling costs
  - identifies a canonical set of cost sources or value activities
- Value chain was adapted into a value chain for software development representative of experience at TRW (see figure)
  - 80% devoted to operations: management (7%), QA-CM (5%), Technical phases (38%), rework (30%)
- Most components of value chain are labor intensive ==> automation should help
- Productivity Improvement Opportunity Tree - derived from value chain analysis.

SOFTWARE DEVELOPMENT VALUE CHAIN [TRW adaptation]



Productivity Improvement Opportunity Tree



- **Get the best from people.** Remember some staffing principles.

  COCOMO shows a factor of

  - 4.18 in productivity difference due to personnel/team capability
  - 2.51 (combined) for experience with applications area, computer system, programming language

- **Make steps more efficient.** Primary leverage factor is the use of software tools to automate the repetitive and labor-intensive portions of each step.

- **Eliminate Steps.** Examples of tools to help reduce/eliminate steps:

  - software analysis tools
  - configuration management tools
  - document generators
  - code generators (automatic programming?)

Personnel Attributes
Staffing Principles [Boehm]

- **Principle of Top Talent.** *Use better and fewer people.*

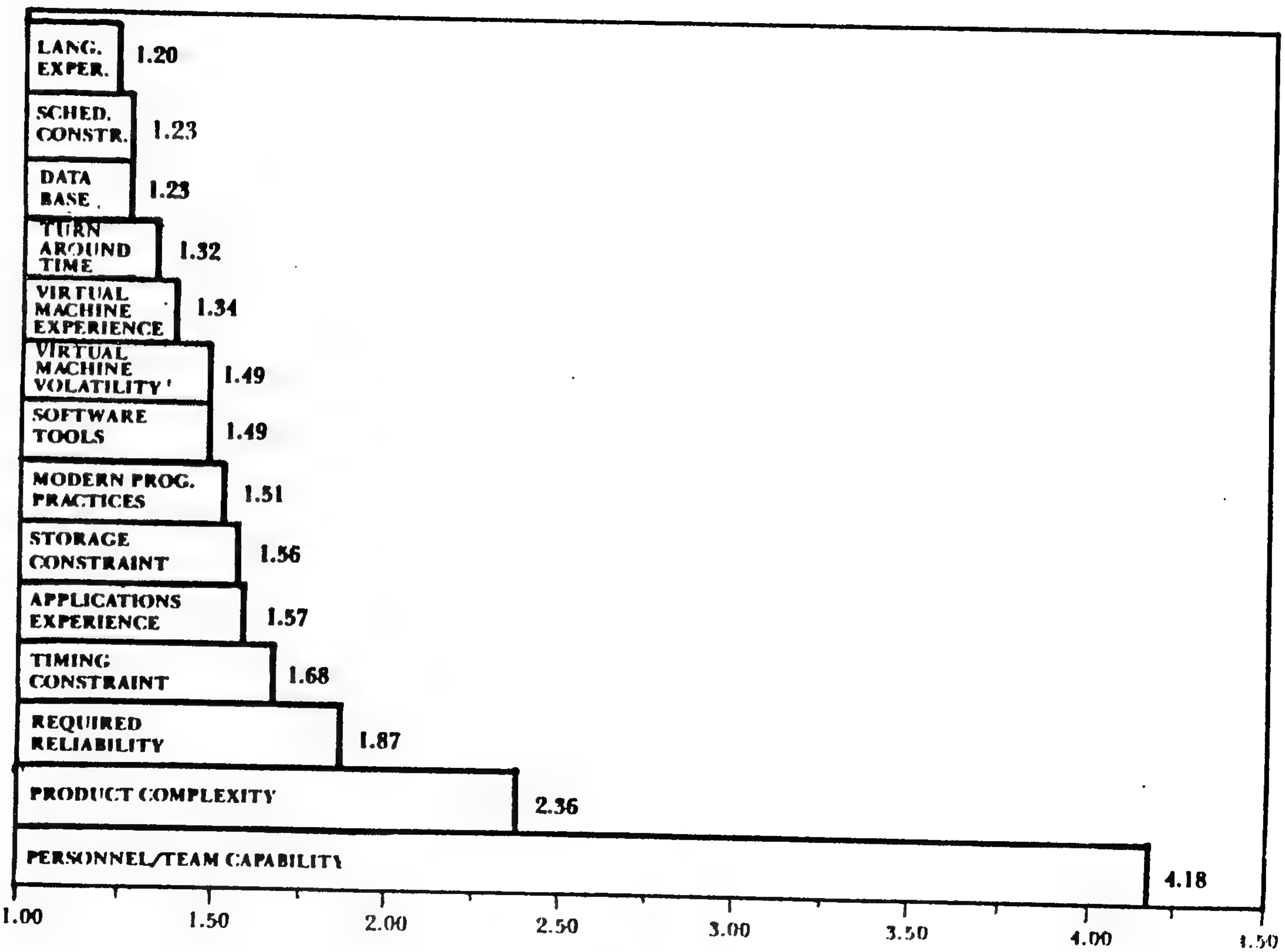
- **Principle of Job Matching.** *Fit the tasks to the skills and motivations of the people available.*

- **Principle of Career Progression.** *Help people to self-actualize.*

- **Principle of Team Balance.** *Select people who will complement and harmonize with each other.*

- **Principle of Phaseout.** *Keeping a misfit in the team doesn't benefit anyone.*

Comparative Software Productivity Ranges
COCOMO



- **Eliminate Rework.**
  - Areas identified:
    - ◇ front-end aids: requirement and design analysis tools, consistency and completeness checkers, automated traceability
    - ◇ knowledge-based assistants - providing intelligent assistance
    - ◇ modern development practices - information hiding/encapsulation, modular design, structured design and programming, walk-throughs, inspections, teams
    - ◇ improved process models
    - ◇ rapid prototyping aids to help understand requirements
  - COCOMO indicates a range of 1.51 during development and up to 1.92 for life-cycle of a large product.

Productivity Improvement Opportunity Tree

- **Build simpler products**
- **Reuse components/(Semi) Automatically Generate Components**
  - libraries of software components: math, statistical, data structures
  - 4GLs, application generators (60% fewer DSI, 60% fewer man hours)
  - Domain-specific architectures and technology
  - Reuse technology

Non-development options

- Biggest cost driver is the number of instructions developed
- Primary controllable factor is the reuse of code via options:
  - Purchase commercial packages
  - Reuse or Adapt in-house software (special case of purchasing)
  - Application Generators
    - ◇ significant cost savings
    - ◇ supports rapid prototyping
    - ◇ but ...
    - ◇ exists mostly in restricted domains
    - ◇ questionable performance of generated code
    - ◇ unclear extensibility

Acquiring Products

- Many types:
  - COTS - Commercial Off The Shelf
  - GOTS - Government Off The Shelf
  - ROTS - Research Off The Shelf
- Major advantages and difficulties in reusing commercial software to be considered in performing a cost-benefit analysis.

Advantages	Difficulties
Cost Savings	In-house Compatibility
Earlier Payback	Procurement Delays
Man-power Savings	In-house Expertise
General Improvements	In-house Improvements
Lower risk	Controllability
Good Documentation	Extensibility
Reliability	

Function Points as a measure of productivity

- Martin and Jones claim that function points (FP) are a good measure.
- Productivity using Function points [courtesy of Dr. James Martin]

Project	FP/Staff month	Cost per FP
Large gov't project	2	1500
Average COBOL	8	1000
3GL w/ prod. aids	13	500
4GL CASE done right	35	250
"Mild" RAD*	100	150
Radical RAD	200	75

* RAD - Rapid Application Development

Function Points

- A measure of complexity widely used for commercial systems.
- Methodology invented by A. Albrecht at IBM in the middle 1970s.
- It measures software by quantifying the functionality provided external to itself (e.g., inputs, outputs, logical data), based primarily on logical design.
- Characteristics:
  - It measures **what** is delivered to the end user, not *how* it is delivered.
  - Geared towards users' understanding
  - Independent of the programming language, technique or technology
  - Reliable early in the design cycle
- Basic Hypothesis: The functions of an application are a more significant determinant of application size than the number of lines of code

Estimated software development costs using full CASE (C. Jones)

"Tables are speculative (1990); however they seem to be realistic estimates of the effect of true automation and displacement of manual labor on software development"

Cost Savings

Application size (in FPs)	Today's costs	with full CASE	Improvement ratios
1	$375	$15	25 to 1
10	$5000	$250	20 to 1
100	$125,000	$8000	15 to 1
1000	$3,000,000	$250,000	12 to 1
10,000	$80,000,000	$10,000,000	8 to 1
100,000	$1,250,000,000	$250,000,000	5 to 1

Time Savings

Application size (in FPs)	Today's schedules	with full CASE	Improvement ratios
1	1.25 days	30 minutes	25 to 1
10	1 month	1 day	20 to 1
100	1 year	1 month	12 to 1
1000	3 years	6 months	6 to 1
10,000	5 years	1 year	5 to 1
100,000	10 years	3 years	3.3 to 1

The Chicken or the Egg Problem:

Who comes first: the methodology or the tool?

Does one buy tools first then squeeze them into a methodology?

Do you hire a methodology consultant and adopt the tools they recommend?

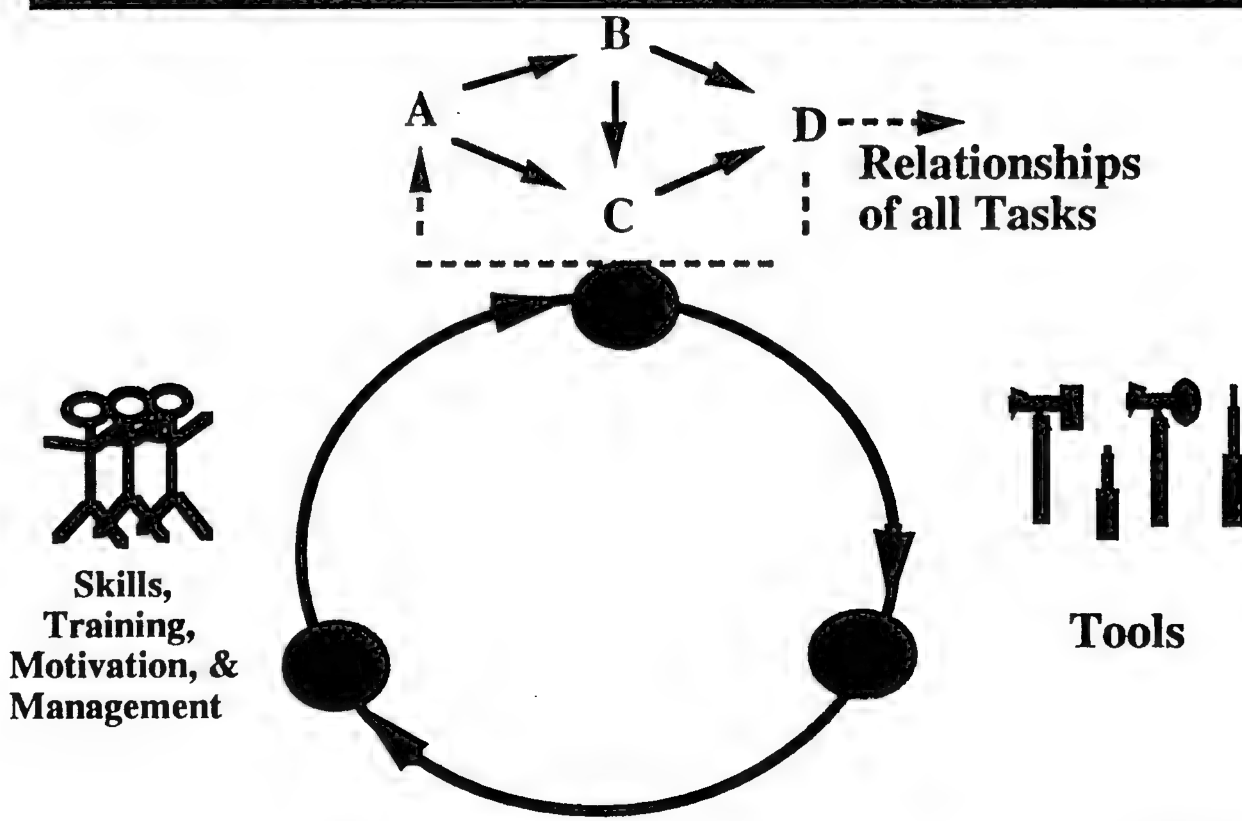
No one knows the clear answer!!!!!

However,

Major productivity improvements cannot occur without a clear marriage between methods, techniques and tools.

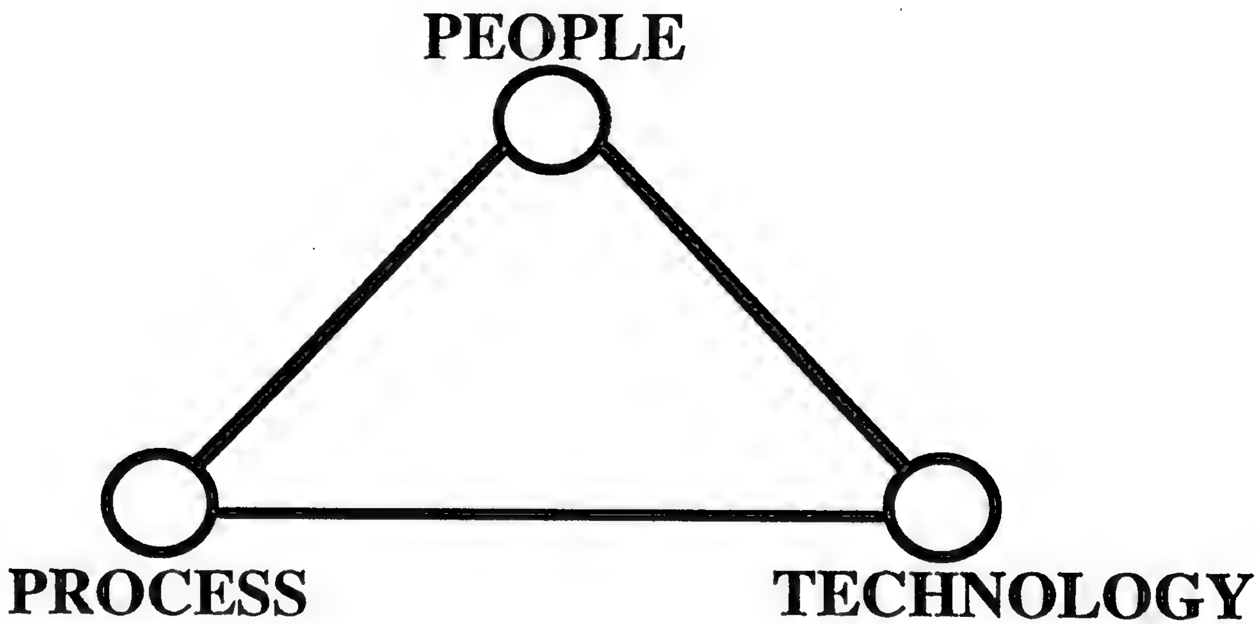
A Definition of Process (SEI)

Set of activities, methods, and practices that guides people (with their software tools) in production of software



[Adapted from SEI, 1991]

Leverage Points for Software Process (SEI)



Major determinants of software cost, schedule, and quality performance

[Adapted from SEI, 1991]

Ingredients of Process Descriptions

Describing a project or an organization's process involves identifying and relating:

- **Project User Roles,** e.g., project manager, developer, configuration manager
- **Activities,** e.g., design, problem reporting, document generation
- **Data (artifacts),** e.g., documents, plans, designs, code, reports
- **Resources** e.g., people, equipment, schedules

Examples of Process Models/Approaches

- Life-cycle Process Models (typically high level)
  - Waterfall
  - Incremental
  - Evolutionary
  - Cleanroom Engineering (IBM)
  - Ada Process Model (TRW)
  - O-O Project DataBase Model (PMDB+)
  - Rapid Development Approach
  - Business Re-engineering
- Sub-processes
  - Configuration Management Process
  - Design Process
  - Review Process
  - Asset certification Process
  - Management Process (e.g., estimating, planning)

Examples of Process Models

- Life-cycle Process Models (typically high level)
  - Waterfall
  - Incremental
  - Evolutionary
  - Cleanroom Engineering (IBM)
  - Ada Process Model (TRW)
- Sub-processes
  - Configuration Management Process
  - Design Process
  - Review Process
  - Asset certification Process
  - Management Process (e.g., estimating, planning)

Waterfall Model

Key characteristics:

- Sequential phase-based approach; reviews at end of each phase
- Products are baselined before next phase begins
- Changes to baselines are controlled.

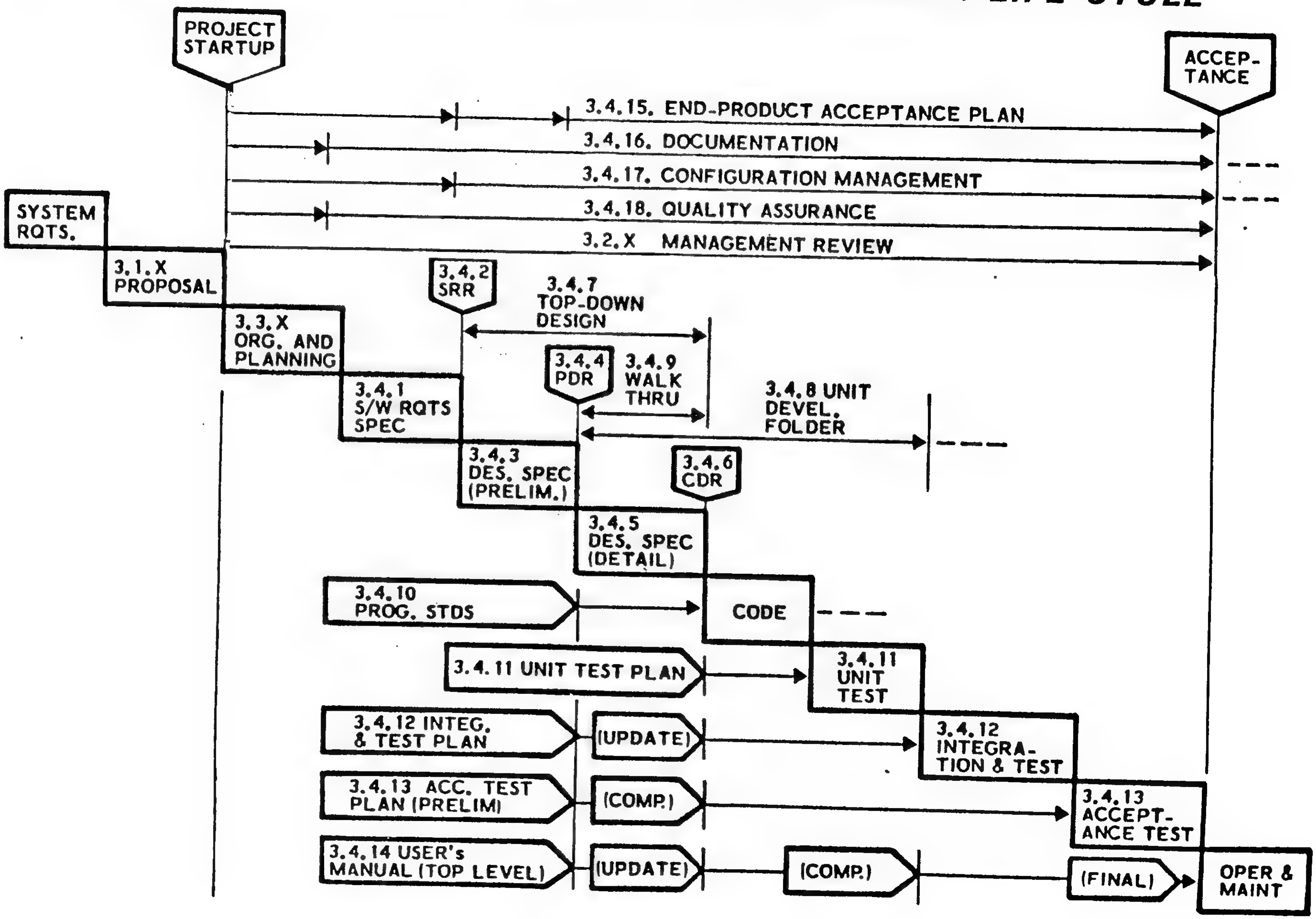
Advantages:

- enforced disciplined approach
- documentation provided at each stage

Disadvantages:

- reality brings iteration
- requirements are not known fully up front
- customer does not see product until end
- too high level

TRW SOFTWARE POLICY COVERS THE SOFTWARE LIFE CYCLE



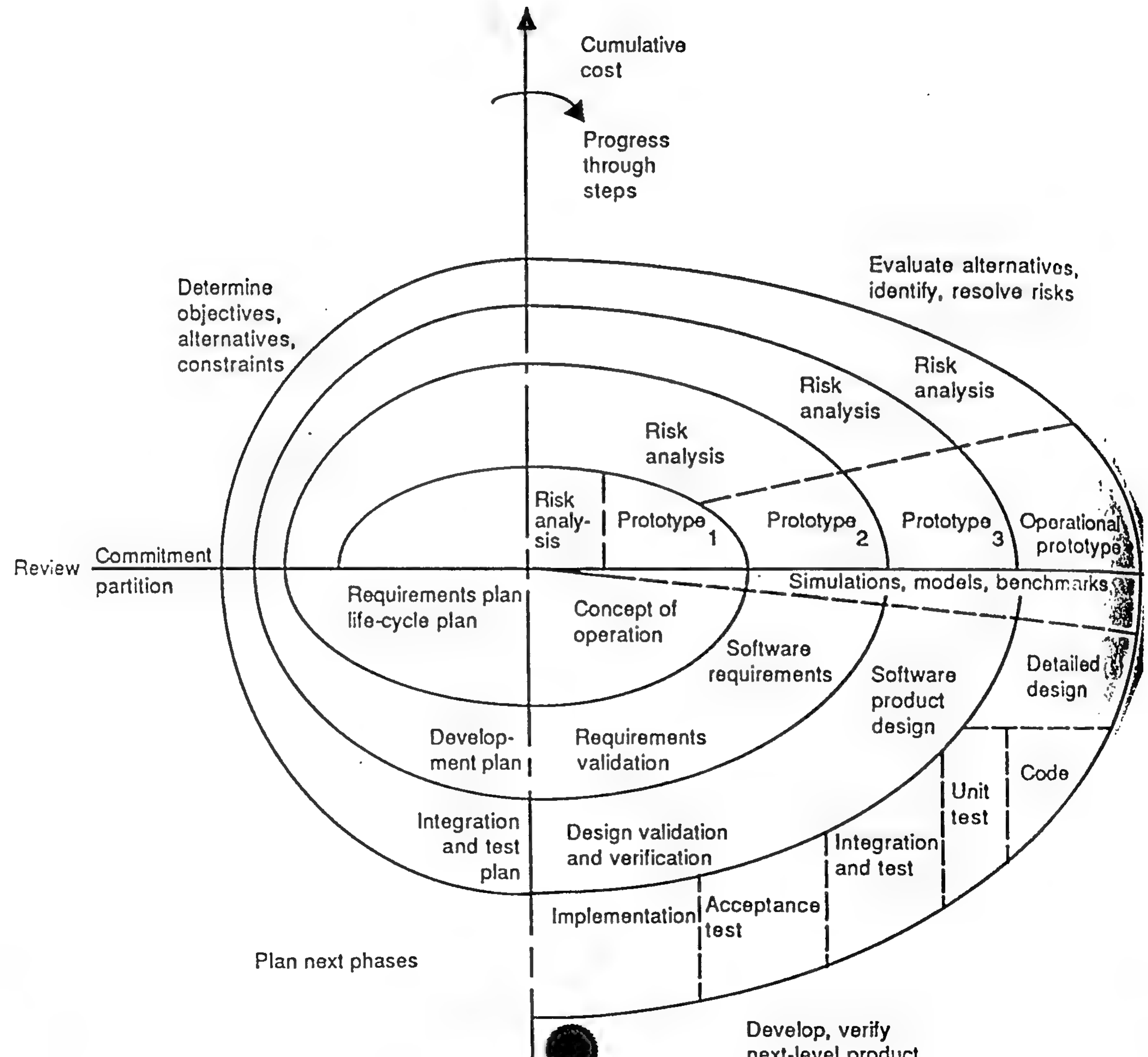
Ada process model [TRW]

Used Evolutionary Strategies and Ada Tools

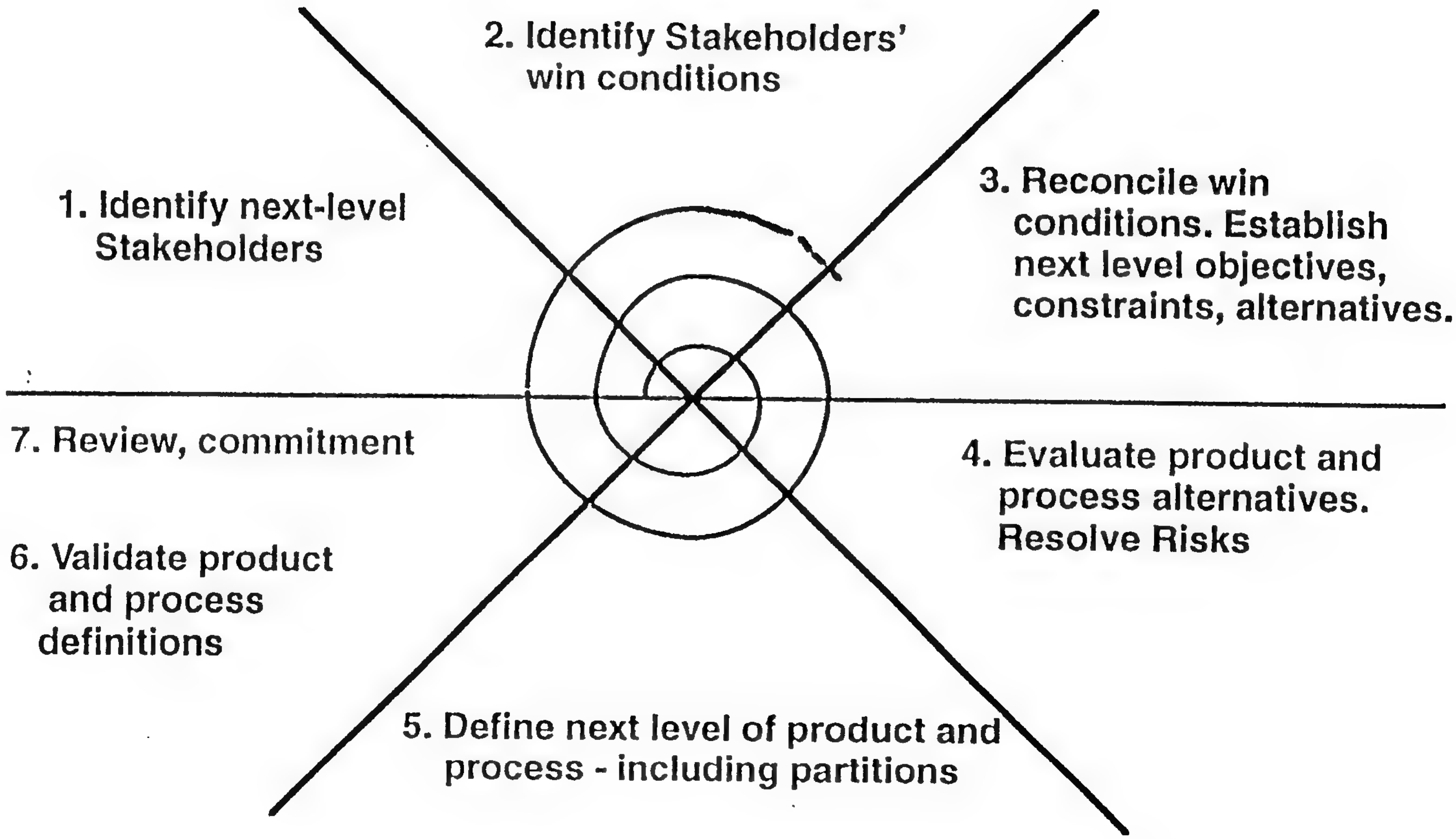
Conventional		New Process Model Strategy
Monolithic Development	==>	Incremental Development
System Execution at end of life-cycle	==>	Early Demonstration
Flowcharts/PDL	==>	Compilable Ada Skeletons early in life-cycle
Documentation based reviews	==>	Demo based Reviews
Specify then build	==>	Continuous Adaptation

Spiral Model

- A framework which accommodates hybrid approaches.
  - Iterative approach based on risk identification and analysis
  - Seems to extend waterfall to include risk analysis
  - Uses prototyping as a means of minimizing risks
  - Explicit but flexible process milestones
- Prescribes Risk Management Approach:
  - Identify project's top 10 risk items (see attached table)
  - Present a plan for resolving each risk item
  - Analyze status and update list of risk items, plan and results monthly
  - Initiate appropriate corrective actions.


Cumulative cost
Progress through steps
Determine objectives, alternatives, constraints
Evaluate alternatives, identify, resolve risks
Risk analysis
Risk analysis
Risk analysis
Risk analy-sis
Prototype 1
Prototype 2
Prototype 3
Operational prototype
Review
Commitment partition
Simulations, models, benchmarks
Requirements plan life-cycle plan
Concept of operation
Software requirements
Software product design
Detailed design
Develop-ment plan
Requirements validation
Code
Unit test
Integration and test plan
Design validation and verification
Integration and test
Implementation
Acceptance test
Plan next phases
Develop, verify next-level product

Theory W Extensions to Spiral Model



Summary of Process Models and Preferred Domains [Boehm]

DOMAIN CHARACTERISTICS	PROCESS MODEL	EXAMPLES
• WELL-UNDERSTOOD APPLICATION • SEVERAL COMMERCIAL PACKAGES AVAILABLE	ACQUIRE COTS	• ELECTRONIC PUBLISHING • BASIC FINANCIAL FUNCTIONS • BASIC INVENTORY CONTROL
• STRONG EVOLUTION-SUPPORT CAPABILITY AVAILABLE: 4GL, TRANSFORM, FORMS MGMT. • APPLICATION FUNCTIONS POORLY UNDERSTOOD • STANDALONE APPLICATION	EVOLUTIONARY DEVELOPMENT	• SMALL-ORGANIZATION MIS • SPREADSHEET APPLICATION • FORMS-PROCESSING APPLICATION
• APPLICATION FUNCTIONS WELL UNDERSTOOD • SYSTEM ARCHITECTURE WELL UNDERSTOOD • NO MAJOR RISK ITEMS • 4GL-TYPE CAPABILITIES INSUFFICIENT	WATERFALL (Ada-ENHANCED)	• MEDIUM-LARGE INVENTORY CONTROL • OFFLINE DATA REDUCTION • SIMPLE MESSAGE HANDLING
• MAJOR ISSUE POORLY UNDERSTOOD • PARTIAL, FRAGILE CAPABILITIES SATISFACTORY	EXPLORATORY RISK REDUCTION	• EXPERT SYSTEMS • PATTERN RECOGNITION • SITUATION ASSESSMENT
• MAJOR ISSUES POORLY UNDERSTOOD • STABLE CAPABILITIES REQUIRED	EXPLORATORY RISK REDUCTION FOLLOWED BY WATERFALL (Ada-ENHANCED)	• LARGE USER-INTENSIVE SYSTEMS — C&C, INTEL, ADVANCED SDE'S • STRONGLY EMBEDDED SYSTEMS • MAJOR REUSE OPPORTUNITIES, UNCERTAINTIES
• DOWNSTREAM FUNCTIONS POORLY UNDERSTOOD • NEED FOR EARLY SUBSET CAPABILITY • MANPOWER CONSTRAINTS • NEED TO STABILIZE REQUIREMENTS DURING DEVELOPMENT	INCREMENTAL DEVELOPMENT	• LARGE USER-INTENSIVE SYSTEMS • VERY LARGE SYSTEMS — CORPORATE MIS, SDI, CORPORATE LOGS. MGMT. • MANPOWER-CONSTRAINED DEVELOPMENTS

CURRENT INTERPRETATION OF "REQUIREMENTS"

COTS-Driven Software Process

- Develop top-level requirements definition and rationale
- Use requirements to determine set of COTS evaluation criteria
- Evaluate COTS candidates with respect to criteria
- Choose best combination of candidates
- Modify requirements to reflect selected COTS capabilities and deficiencies.

PMDB+ model [Penedo/Shu]

Object-oriented model of the life-cycle

- **object types**: Requirement, Person, Change Item, Milestone, Software Component, Task, Test Procedure, Problem Report
- **attributes**
- **relationships**
- **operations associated with objects**, e.g., Problem Report: open, close, disposition; Software Component: define, decompose, analyze, build, baseline, check out, test, etc.
- **active data** - actions which occur as a consequence of changes to the system.
    - ◇ Example: changes to estimated # lines of code of a software sub-component causes changes to # lines of code for parent component.

Example of PMDB+ Component

Object Type: Problem Report

Attributes	*Relationships*	*Operations*
Problem Report ID	Identified by: Person	Close
Title	Controlled by: Person	Display
Type	Solved by: Person	Disposition
Date Opened	Approved by: Person	Open
Date Closed	Causes: Change Item	Print
Severity	Affects: Document	Update Description
Problem Description	Software Component	Update Solution
Disposition	Requirement	
Proposed Solution	Test Case	
Solution Disposition		
Effect		
Approvals		
Status		

Rapid Application Development (RAD) Approach

- Advocated by J. Martin - used in commercial/business arena
- Claimed as a way of putting together: tools, people, methodology, management.
- Characteristics:
  - User involvement early
  - Prototyping using 4GLs and CASE tools
  - Using code generators
- Also encouraging use of JRP (Joint Requirement Planning) and JAD (Joint Application Development)
- Phases are:
  - Requirement Planning
  - User Design
  - Construction
  - Cutover Phase

JAD sessions

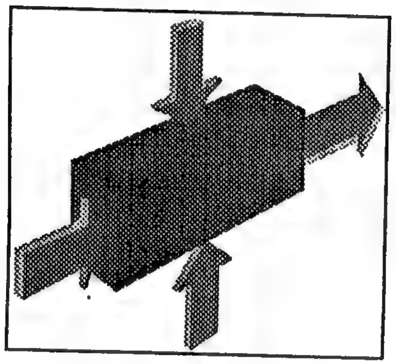
- Joint Application Development
- Design workshops - sometimes more than one are needed
- People involved
  - conducted by session leader
  - teams should consist of 2-4 people
  - one scribe
  - business people and users involved in design
- Involves: paper, CASE tools and prototypes
- Products:
  - screen and report design
  - documentation in repository
  - built prototype

Business Re-Engineering

- Deals with process improvement within an organization.
- Def. 1: The radical reshaping of business processes, structures, management systems, and values to achieve quantum leaps in performance.
- Def. 2: The fundamental analysis and radical redesign of critical business processes to achieve dramatic improvements in cost, quality and speed. [Hammer]
- Def. 3: The process of overhauling archaic, inefficient organizational structures. [Hammer]
- A process in itself:
  - Work is defined and controlled by "business processes" rather than by functional organizations
  - The language is the language of business models

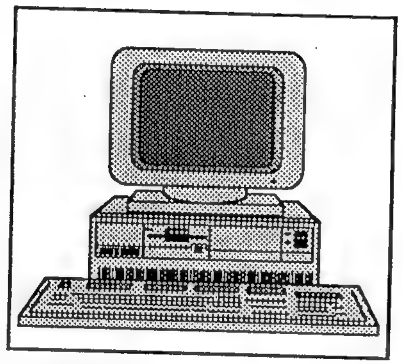
Dimensions of Change

Processes



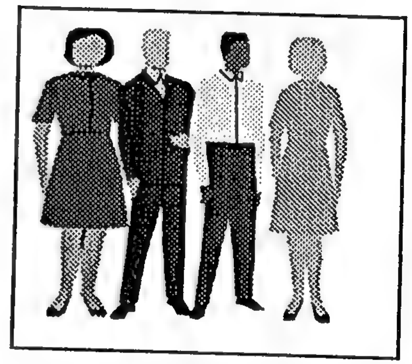
- Usefulness and Efficiency of Current Process
- Automation of Necessary Work Processes

Technology



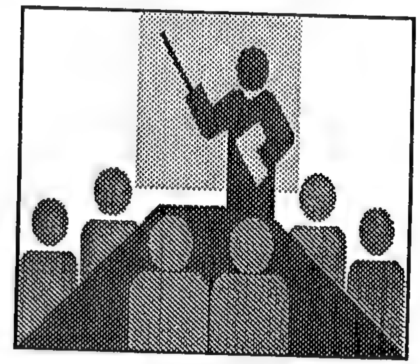
- Avoidance of Redundancies & Incompatibilities
- Technology Supporting Organization's Mission

People & Culture



- Skill Base to Build New Capabilities
- Readiness to Use New Technology

Organization & Structure



- Flexibility for Changing Responsibilities

Price Waterhouse

Business Re-Engineering (Cont.)

- Essential ingredients:
  - Commitment to change
  - Organizational acceptance of a disciplined redesign approach
  - Linkage to mission, goals and objectives
  - Top management and organization-wide involvement
  - Customer-focused product or service
  - Results-driven processes
- Examples of methods/tools
  - IDEF0 - Activity Modeling (a revision of the SADT methodology). Models business activities and their inputs, outputs, controls and mechanisms.
  - IDEF3 - Work flow Modeling. Models work flows that associate business activities.

STARS' Levels of Process Automation Adapted

- Manual or Document-driven (paper based)
- Document-driven with some on-line support
- Tool support
- Process partially integrated with tools - user controlled
- Pro-active process-based SEE support (guidance and enforcement)

Motivation for the Definition of Software Processes

[adapted from P. Garg (HP)]

- **Guidance.** What needs to be done next?
- **Quality.** Have all the needed steps been carried out?
- **Analysis.** Are we doing the right things?
- **Automation.** Can some of the routine work be automated?
- **Coordination.** Who depends on who's work?
- **Communication.** Who needs to inform about their work to whom?
- **Measurement.** What can and should be measured?
- **Recording.** What needs to be preserved?

CASE vs SEEs

- CASE: Computer Aided (Assisted, Automated) Systems (Software) Engineering
- SEE: Software Engineering Environments
- PSEE: Process-driven (or process-based) SEE

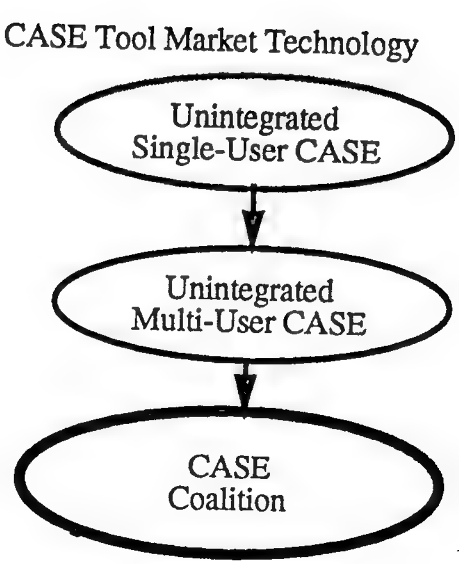
SEE Definitions:

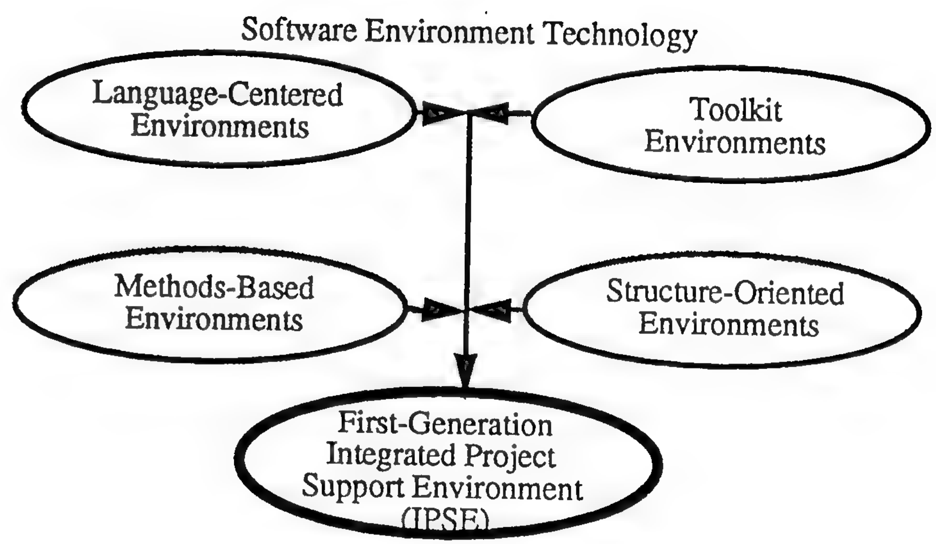
- Independent tools (single or multiple platforms)
- Integrated tool-kits
- CASE coalitions
- Process-driven, i.e.,
  - ◇ Process encodings drive its execution
  - ◇ User interaction is process based

CASE Definitions:

- The disciplined and structured engineering approach to software and systems development.
- The use of computer and software technology for improving the development process and the products of that process
- Tools and methods to support an engineering approach to system and software development at all stages of the process.

CASE vs SEEs


CASE Tool Market Technology
Unintegrated Single-User CASE
Unintegrated Multi-User CASE
CASE Coalition


Software Environment Technology
Language-Centered Environments
Toolkit Environments
Methods-Based Environments
Structure-Oriented Environments
First-Generation Integrated Project Support Environment (IPSE)

First Generation CASE

- Language-Support Environments (C, Ada, C++)
  - Editor
  - Compiler
  - Debugger
  - Analyzers
- Automated Support for Structured Analysis and Design
  - Data Flow Diagram Editor
  - Structure Chart Editor
  - Entity-Relationship Diagram Editor
  - Data Dictionary
- User Interface Prototyping
  - Screen Painter
  - Report Generator
- Re-invented CASE
  - Program Generators
  - Library Utilities
  - Documentation Tools

More recent CASE Tools

- Visual Interface (GUI)
- Repository support
- Object-oriented methodology support
- OODBMS
- Mixed Product environments/initial tool integration
  - Design and document generation
  - Database and GUI
- Design-to-code generation
- Work-flow
- Process Definition
- Client-server support
- Metrics collection
- Group support
- Multi-media

Problems/Complaints

- **CASE tools:**
  - buggy, costly, slow
  - non-scalable, i.e., only supports small projects
  - non-portable
  - not integrated with other tools
  - hard to learn
  - forces too much detail per phase of use
- **Problems with Current SEEs**
  - Some good solutions (more like CASE capabilities), for limited portions of life-cycle activities
  - Project customization is difficult and costly
  - Little uniformity across activities
  - Compatibility between independently developed tools
  - Little effective support for process, reuse or COTS integration

Benefits of CASE Tools

- Automation of manual work
- Enforcement of standardized notation and methods among project users
- Syntactic and semantic checking support consistency and completeness of designs, document and code
- Easy access to definitions and data
- Help with automated documentation

Failures of CASE

Internal and external feedback:

- Only 20% of CASE product licenses are actually in use
- Few features utilized within tools
- There exists little integration among tools
- Platform dependencies cause problems
- Inadequate tool training provided
- Lack of adequate time for learning curve
- Lack of robustness and scalability

CASE - Changing the culture [Yourdon, McClure]

- Specification-driven process rather than a code-driven process
- Emphasis on building systems from reusable parts or in a generative mode (documentation, code) with continual automated checking
- Engineers will work in a responsive, interactive environment, with a sophisticated human interface
- CASE environment will incorporate expert system technology in such areas as methodology selection and usage.
- Culture change will be accompanied by a dramatic improved environment for software engineers, with the following change in characteristics
  - user-friendly interface => customized
  - diagnostic => corrective
  - supportive => directive
  - helpful => teaching
  - responsive when queried => reacts to events it detects
- The most difficult task of the software organization will be that of striking a balance between adapting tools to the way people work and adapting people's work habits to the constraints and capabilities of the tools.

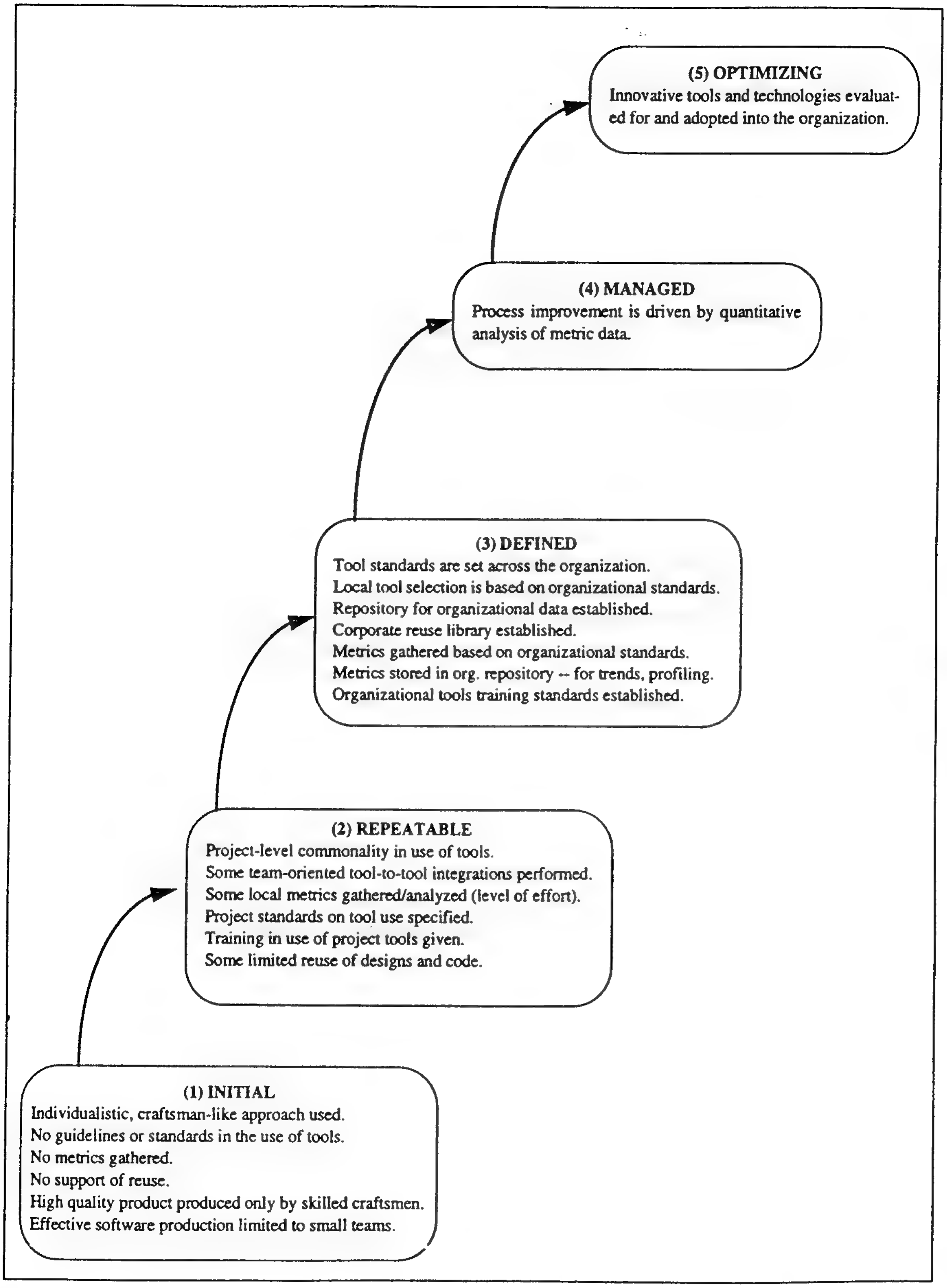


Figure 2-2: Tool-Use Characteristics at the Five Maturity Levels

: "A practical guide to the technology and adoption of Software Process Automation." [Alan Christie]

Outline

Introduction

- Rationale - Cost and Productivity Highlights
- Process - concepts, examples
- CASE/SEEs - concepts

Models, Characterizations and Examples

- **Concepts**
- **Reference Models**
- **SEE architectures types and Examples**
- **Integration**
- **Lessons Learned**

Conclusions

Architecture

- Buzzword of the 80s; being re-discovered in the 90s
- Imprecise, broad, sometimes contradictory definitions
- Unclear distinctions between System and Software Architectures
- Unclear distinctions between Architecture and Design
- Definition [Webster]: The art or science of designing/building composite wholes.
- Software Architectures should convey information about [PSEEA Workshop]:
  - components
  - static inter-relationship of components
  - dynamic interactions of components
  - properties/characteristics
  - constraints on the items above.
- "There is not an architecture, but a set of architectural representations which are additive, complementary" [Zachman].

Architecture - Shifting Views

[extension of G. Fox - TRW]

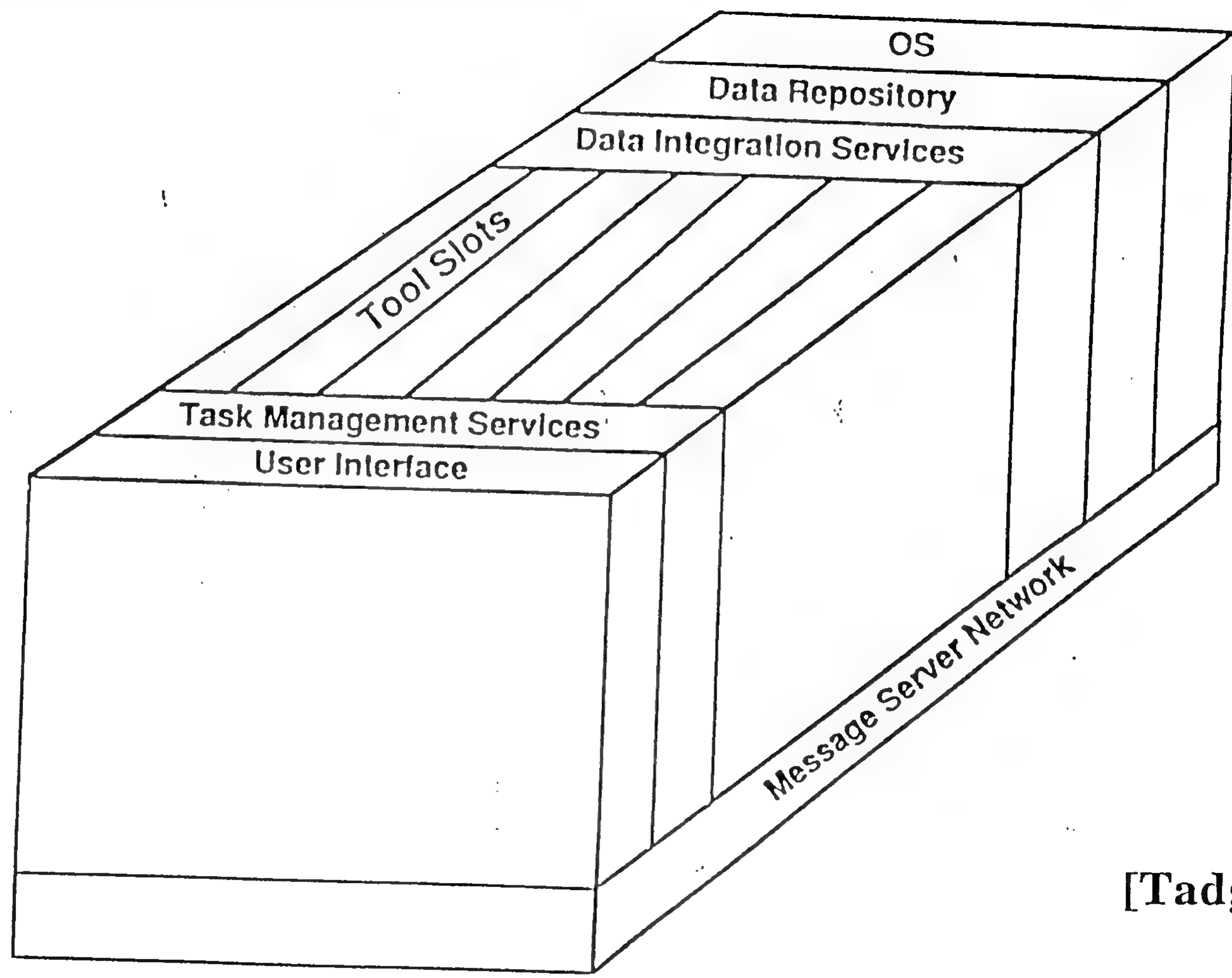
- Traditional Engineering Views of Architecture
  - Bridge between requirements and design
  - Conceptual View from which many systems or products built (e.g., OSI 7-level communications model)
  - Functional or structural description of system
    - ◇ Module Interconnection Languages/Systems
    - ◇ Functional and Data Flows for software
    - ◇ High-level wiring diagram for hardware
- Emerging view of Architectures
  - Architecture as a continuum that encompasses design
  - Technical management tool for system development and evolution
  - Some proponents:
    - ◇ DoD (Domain Specific Architecture Program (DSSA))
    - ◇ Zackman (late 1980's IBM)
    - ◇ Martin (Information Engineering)

Reference Models (RM)

- Conceptual frameworks for understanding areas of work.
- Sometimes serve as basis of evaluation techniques.
- In the SEE arena, being used as a **conceptual and functional framework** for discussing, presenting and comparing SEEs and their architectures.
- RMs are **not** architectures
- Examples:
  - Toaster/CASE 89 (only a picture)
  - NIST/ECMA (a report - SEE frameworks)
  - PSE RM (full SEE functionality)
  - TRW's CEARM (full SEE)

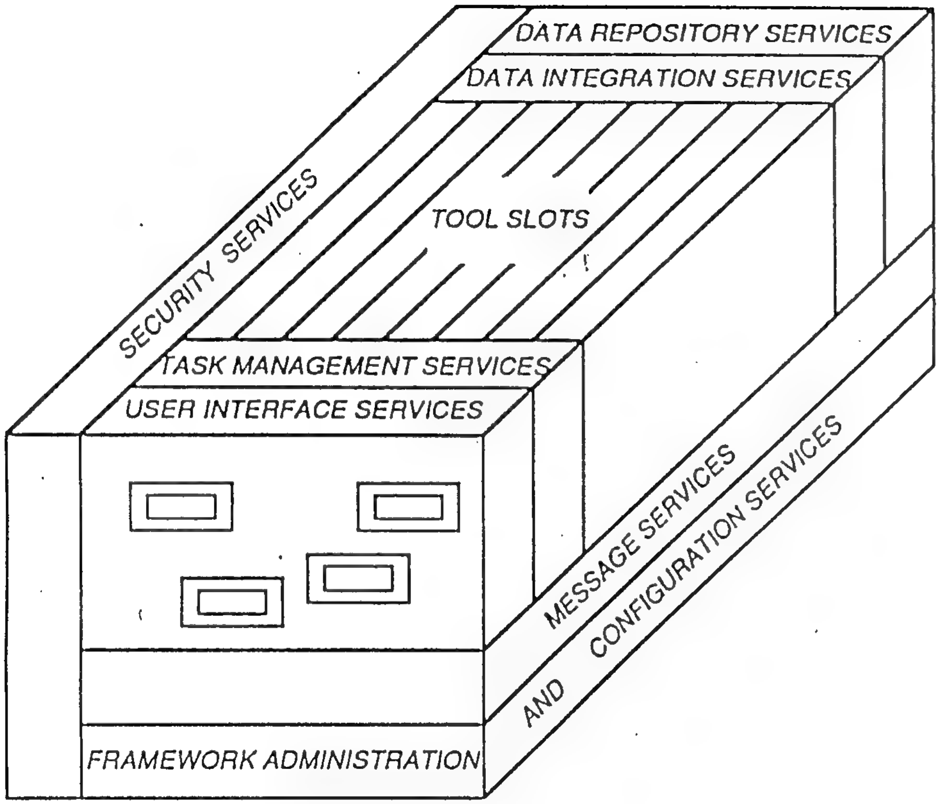
CASE 89 Reference Model

[aka, Toaster Model, ECMA 1990]

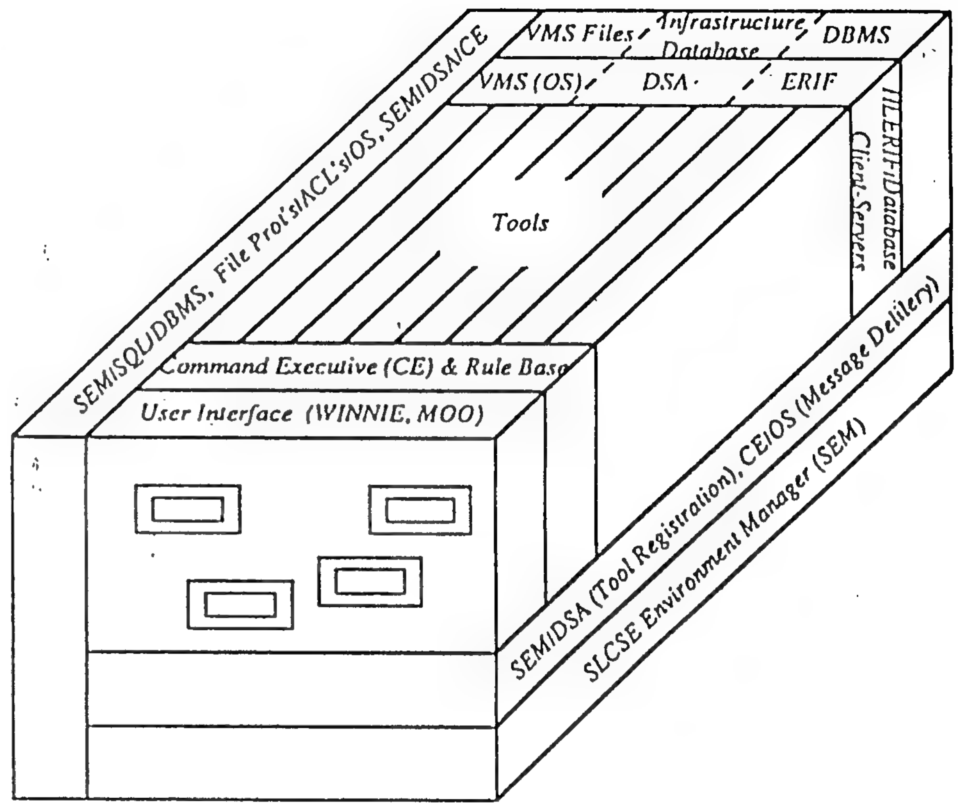


[Tadge, HP]

SLCSE in terms of Case 89 RM



CASE Environment Frameworks
Reference Model



SLCSE Framework

NIST RM

- *Reference Model (RM) for Frameworks of Software Engineering Environments*
- A service-based model for describing SEE frameworks
- Developed jointly by ECMA TC33/TGRM WG and NIST ISEE WG
- Published as a NIST Special Publication 500-211 and ECMA TR/55 3rd Edition.
- M. Penedo and H. Hart were part of the team who developed it
- Currently in wide use in the community as a way of describing SEE frameworks
- Sometimes wrongly called as the Toaster Model (It was actually an evolution and further refinement of the original HP Toaster Model)
- Many SEE (sub) frameworks mapped to it.

NIST/ECMA RM

- It provides a taxonomy of services
- Main groupings are:
  - Object Management Services
  - Process Management Services
  - Communication Services
  - User interface Services
  - Policy Enforcement Services
  - Framework Administration Services
  - Operating System Services
- Each grouping has a collection of services.
- Each service is described via different dimensions.

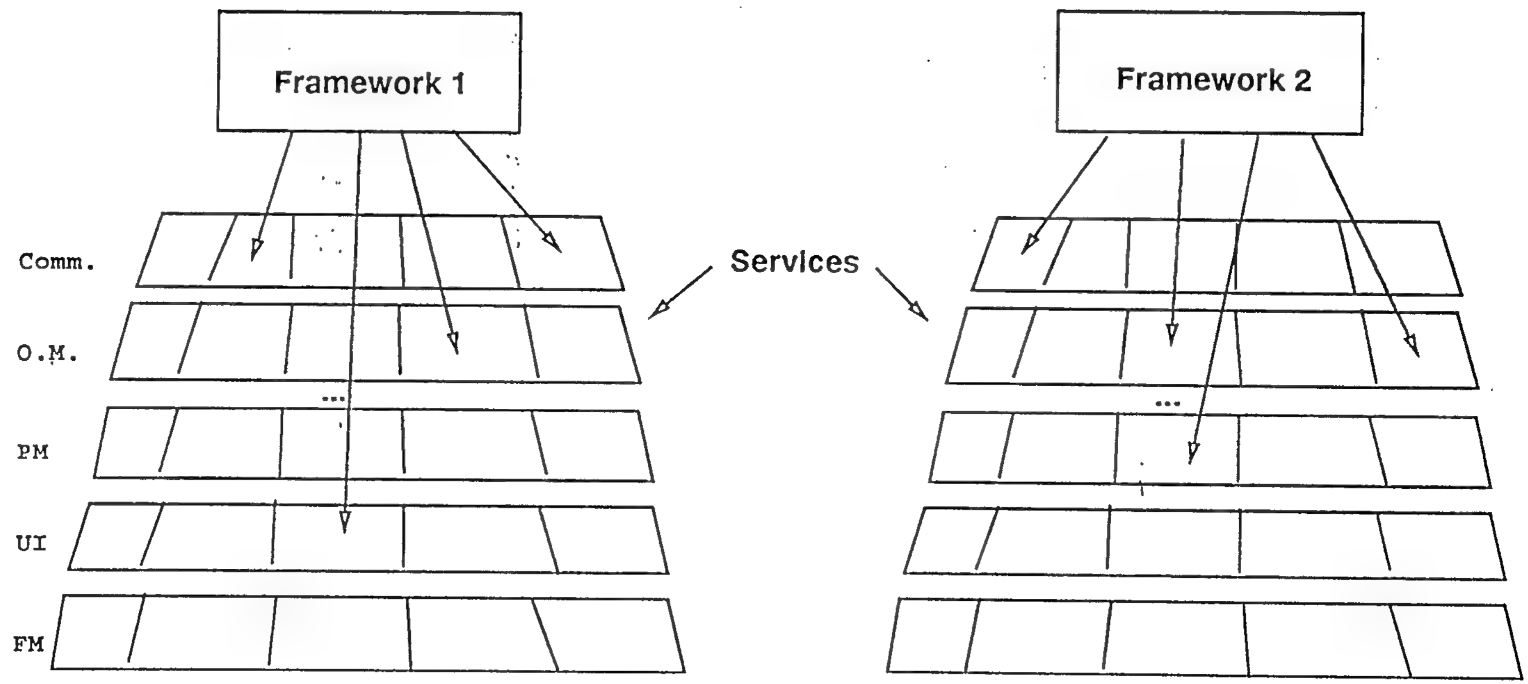
Object Management Services
NIST RM

- Metadata Service
- Data Storage Service and Persistence Service
- Relationship Service
- Name Service
- Distribution and Location Service
- Data Transaction Service
- Concurrency Service
- OS Process Support Service
- Archive Service
- Backup Service
- Derivation Service
- Replication and Synchronization Service
- Access Control and Security Service, ...

Process Management Services

- Process Development Service
- Process Enactment Service
- Process Visibility Service
- Process Monitoring Service
- Process Transaction Service
- Process Resource Service

NIST/ECMA RM for SEE Frameworks (1993)



Service Dimensions - NIST RM

- Each service has a set of asociated dimensions.
- When describing a system, the dimensions describe, from multiple perspectives, how specific system components fulfill/implement that service.
- NIST/ECMA Dimensions for each Service:
  - Conceptual - semantics of service
  - Operations - set of functional operations that implement that service
  - Rule - constraints on implementation of service
  - Types - types of objects used by implementation of service
  - External - how service is accessed
  - Internal - internal implementation details
  - Related services - interactions or dependencies across implementation of services.
  - Examples

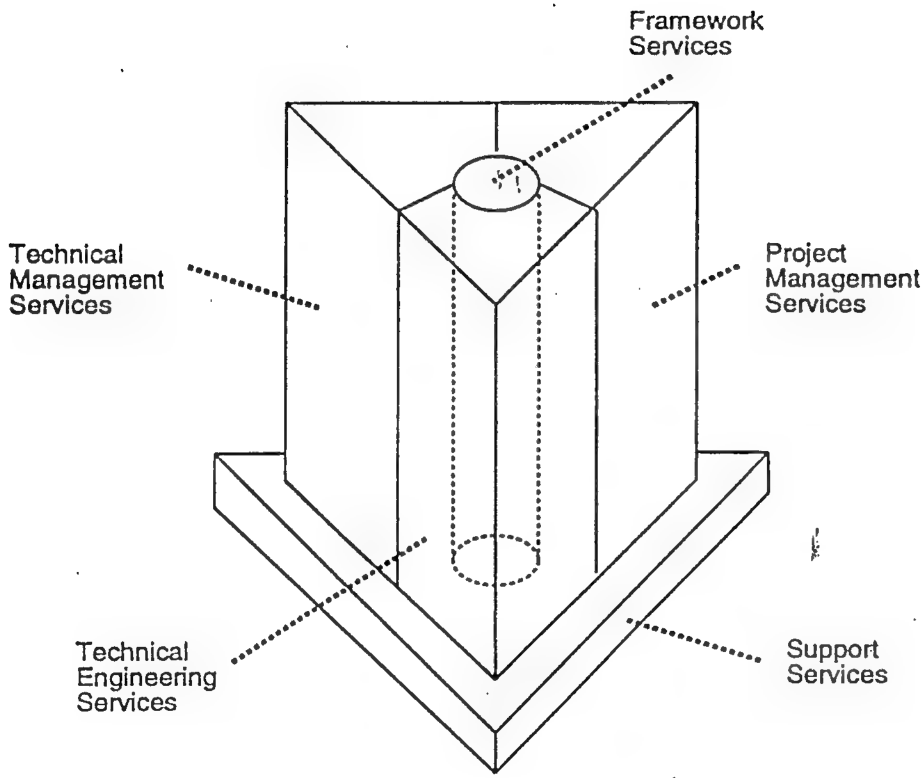
Project Support Environment (PSE) Reference Model

- Developed as part of the Next Generation Computer Resources (NGCR) program - U.S. Navy
- Goal: extend service-based NIST RM to support a full SEE
- Adopted NIST RM for framework
- Defined end-user services
- End-user services are:
  - Technical engineering
  - Technical Management
  - Project Management
  - Support Services
- Latest version published September 1993 by NIST

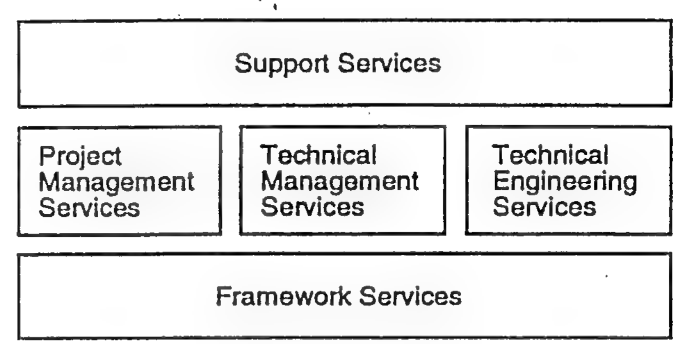
PSE RM (Cont.)

- Technical Engineering Services
  - System Engineering Service
  - Software Engineering Service
  - Life-cycle Process Engineering Service
- Technical Management Services
  - Configuration Management Service
  - Change Management Service
  - Reuse Management Service
  - Metrics Service
- Project Management Services
  - Scheduling Service
  - Estimation Service
  - Risk Analysis Service
  - Tracking Service
- Support Services: ...

PSES RM Depiction



An Illustration of Service Groups

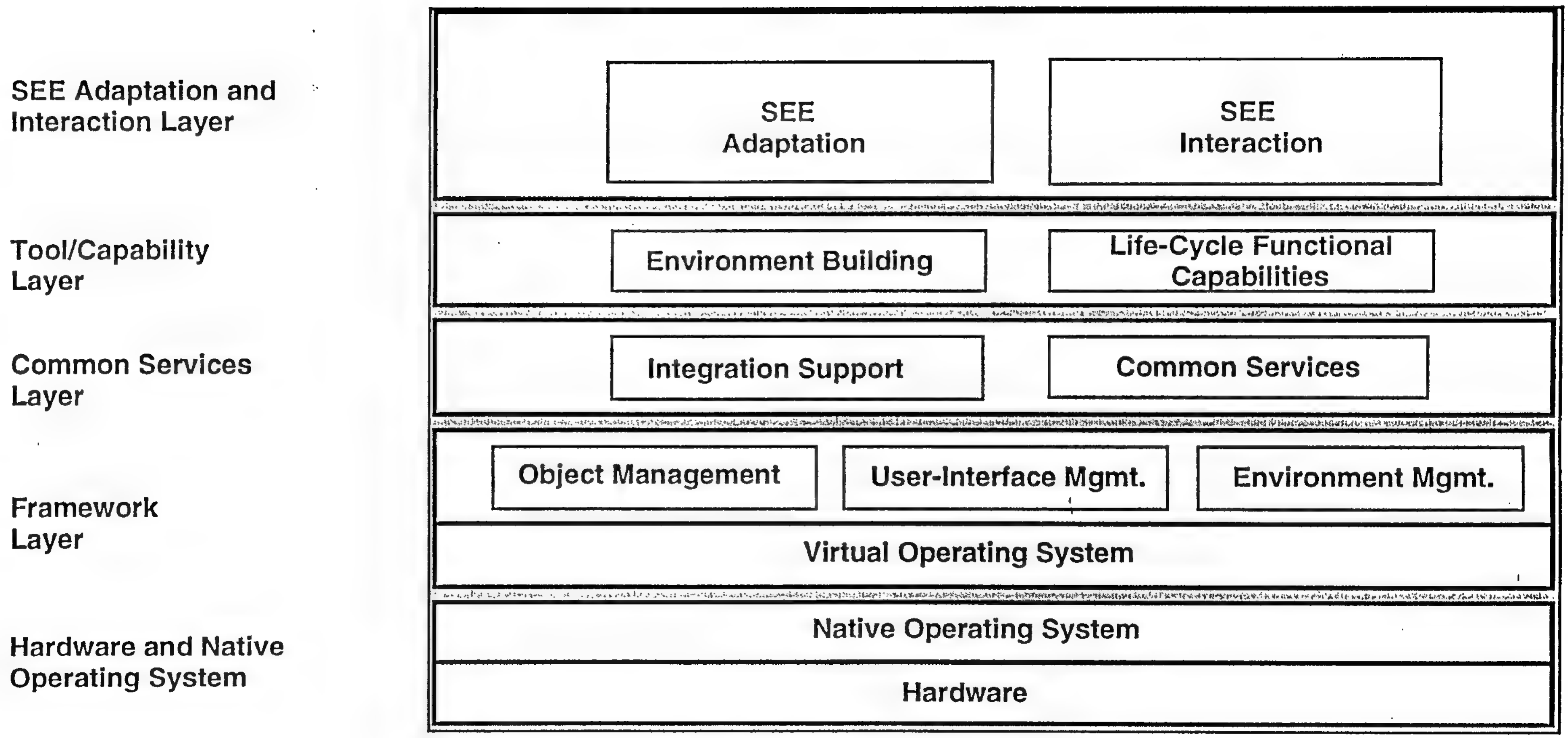


Another Illustration of Service Groups

CEARM Reference Model

- Conceptual Environment Architecture Reference Model (CEARM)
- Developed at TRW by Penedo, Shu and Karrer
- Objective: Support description and comparison of existing sytems.
- Evolution: Work merged with NIST and PSE work.
- Surveyed and mapped systems to RM:
  - Arcadia
  - HP Softbench
  - East
  - Kernel/2r
  - Kernel/1
  - ATIS
  - SLCSE
- Captured lessons learned

CEARM Diagram



[Penedo - TRW]

SEEs: Models, Characterizations and Examples
Outline

- Concepts
- Reference Models
- **SEE Architectures and Examples**
- Integration
- Lessons Learned

Architecture

- **Definition** [Webster]: The art or science of designing/building composite wholes

- A software system's architecture identifies
  - its components,
  - their static inter-relationship and
  - their dynamic interactions.
  - properties/characteristics
  - constraints on the items above

- There are many kinds of and ways of describing environment architectures

Examples of Architectural Components

- Hardware
- Software:
  - Operating System
  - Communication Manager
  - Database/Knowledge-base Manager
  - Tools/Life-cycle components
  - Environment Manager
  - Process Engine
  - Integration Mechanisms
  - ...

Example of Family of Integrated Tools

- IDE - Interactive Development Environment Software Through Pictures (StP) is a suite of integrated analysis and design tools, built on a common core architecture using a common central repository.
- Elements of the family:
  - StP/SE (Structured Environment) - for structured analysis and design, supporting the most popular structured methodologies
  - StP/IM (Information Modeling) - for information modeling to help analyze corporate data and build its RDBMS
  - StP/OMT (Object Modeling Technique) - for object-oriented development, supporting Rumbaugh's OMT methodology.
  - Ada Development Environment, which includes: Object-Oriented Structured Design/Ada, Code Generator for Ada, Design Generator (reverse engineering) for Ada.
  - C Development Environment, which includes tools for: analysis, programming, document publishing, testing, reverse engineering, configuration management.

Examples of PSEEs or PSEEs components

- **PCTE** is an ISO "interface" standard providing object management and Operating System like services supporting portability of applications.
- **PACT** is an environment built on PCTE by an European consortium providing a layer of common services and several tools sets to support software development.
- **EAST** (The European Advanced Software Technology)-Environment is a commercial PCTE-based environment under development providing integration services and tools to support the development of software. Product of the EAST EUREKA Project which is a international consortium of companies including SFGL (France), NOKIA (Finland), DMR (Canada), BULL, DATAMAT, and INTECS (Italy).
- **HP BMS** (Broadcast Message Server) is HP's control integration mechanism. It uses a broadcast paradigm where tools can communicate requests for action or notify the completion of actions independent of each other.
- **HP Softbench** is a commercial SEE consisting of an integrated set of tools for software development built on a message-based control integration set of services (e.g., BMS, Encapsulator)

Examples of PSEEs or PSEEs components

- **DCE** (Distributed Computing Environment), an OSF standard supporting distribution and heterogeneity. It is C based and it includes services for: file managmeent, naming, threads, rpc, timing, security. It allows computers from multiple vendors to work together transparently and share computing power, data and devices.

- **ORB** (Object Request Broker) provides mechanisms by which objects transparently make requests and receive responses, with the objective of providing interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnecting multiple object systems.

- **CORBA** (Common Object Request Broker Architecture) and is a client-server like standard laid down by the Object Management Group (OMG) to define a technology for ORBs.

  CORBA is like RPC except the subprogram name is a server object operation, and the first parameter in each call is an object-id, which is used to locate the server object. Adopted form a joint proposal by: DEC, HP, HyperDesk Corp., NDR, Object Design, SunSoft.

- **OLE** Microsoft's CORBA-correspondent. It was designed to allow users to integrate separate COTS applications as large granularity components, which behave to the user as one system.

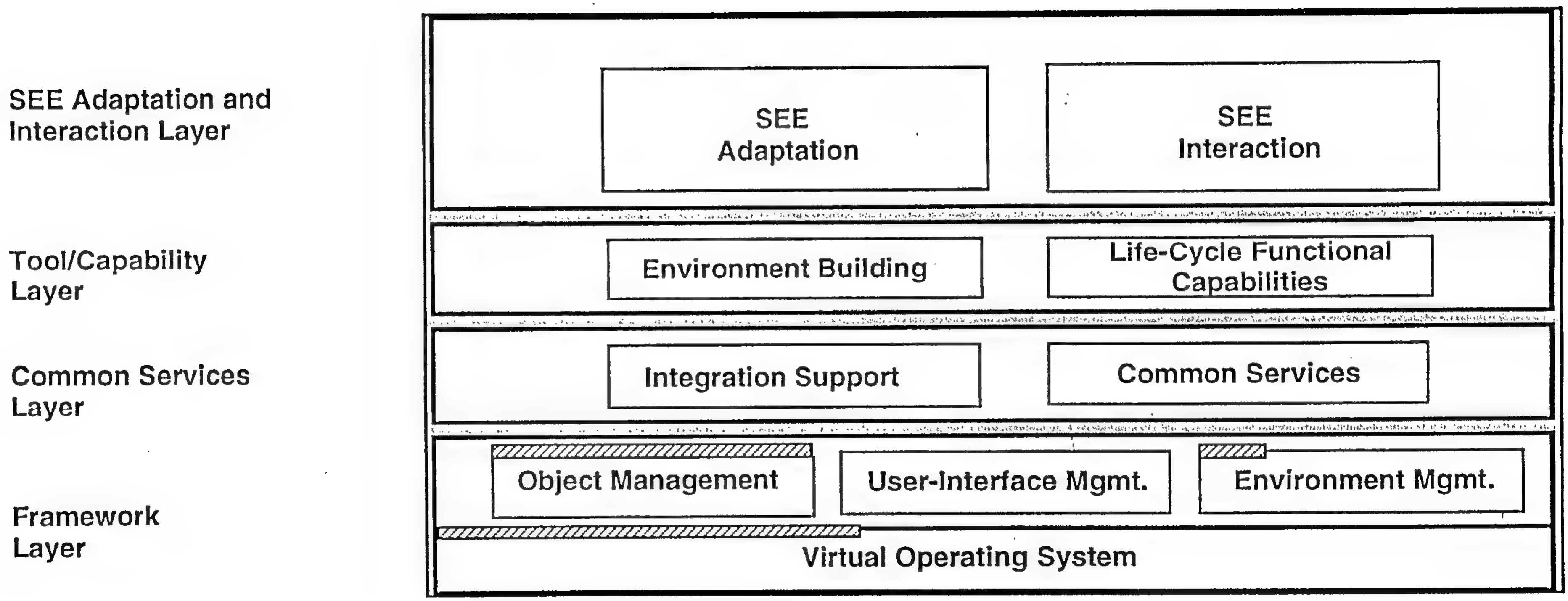
Examples of PSEEs or PSEEs components

- **SLCSE** is a prototype of a software development environment developed by General Research Corporation for the Air Force. Currently being redesigned for commercial purposed by ISSI into Pro-SLCSE to be based on PCTE and to incorporate object oriented and process modeling capabilities.
- **Kernel/2r** (a.k.a. K/2r) is a PSEE research prototype, developed at the University of Dortmund (UniDo) in Germany, which provides an bus-based integration infrastructure for advanced, distributed, and heterogeneous components. It was sponsored by the Eureka Software Factory (ESF) program.
- **Kernel/1** is the commercial counterpart of Kernel/2r within the ESF program; a software-bus based system, which includes the Process Weaver Tool component. Kernel/1 was built by three industrial partners: Cap Gemini Innovation, CAP debis GEI and Sema Group.
- **Arcadia** environment provides a combination of technology and prototype components in support of PSEE construction and in support of the life-cycle. It has been developed by the Arcadia Consortium under ARPA sponsorship.
- **Oz** is a research PSEE prototype which provides capabilities for process modeling and enactment of multiple, heterogeneous, autonomous processes. Latest research is incorporating multiple user collaboration (CSCW) components into the environment. It is being developed at Columbia University in the USA.

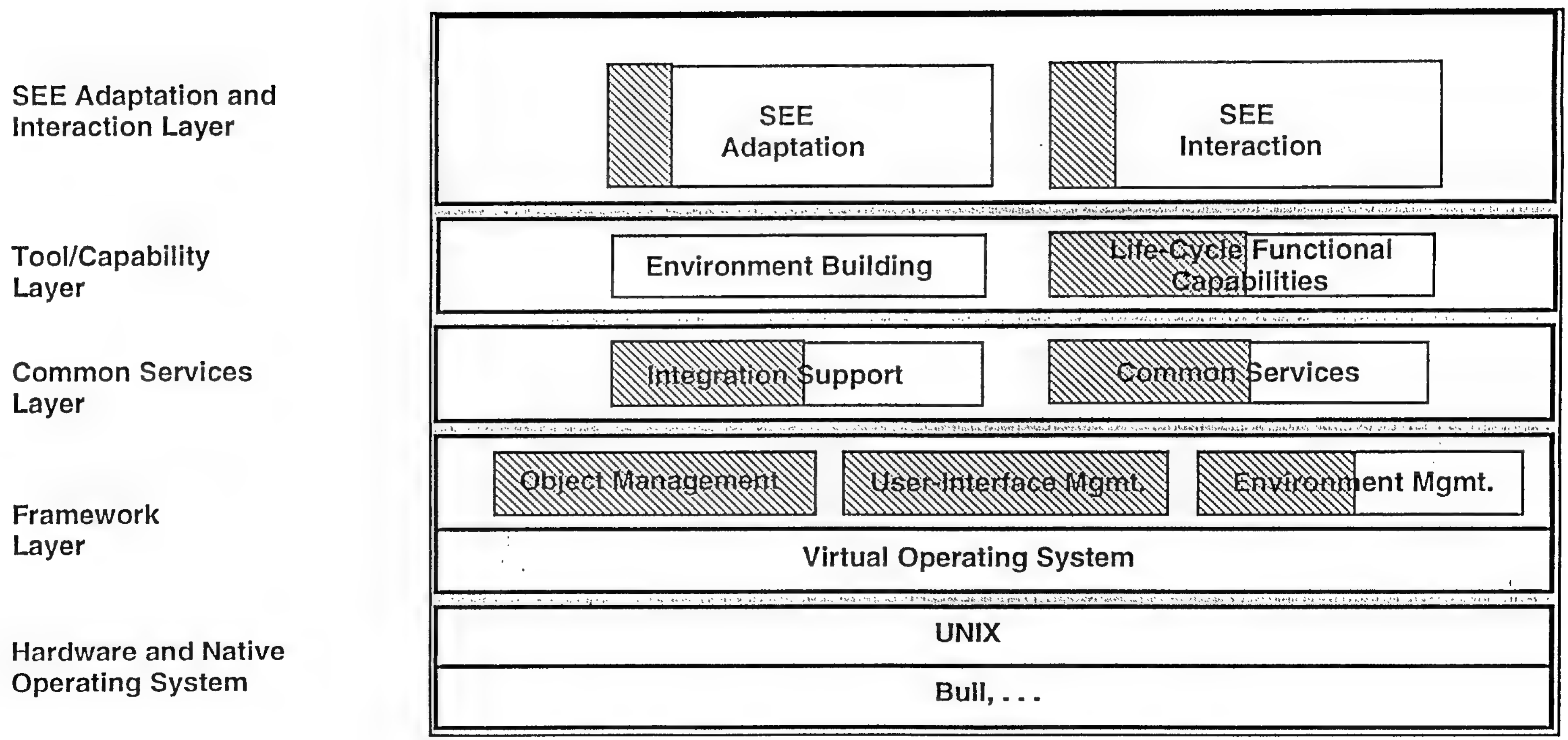
Commercial Process modeling/enactment tools

- **Synervision** is a process enactment tool (with limited support for process modeling) being sold by HP which interacts with other tools via the BMS component.. The representation model is a hierarchy of tasks (a work breakdown structure) having attributes and whose visibility is shared by a work group. It is being sold by HP.
- **LEU** (LION Engineering Environment) is a recent commercial product in Germany, supporting the development of application software on the basis of workflow management technology. It is an evolution of parts of the Kernel/2r environment.
- **Process Weaver** is a commercial product in support of process modeling and execution of life-cycle activities in accordance to the defined process. It supports a Petri-net approach of process modeling and it supports an agenda-based user interaction paradigm. It evolved from the ESF work. Being marketed by CAP Gemini Innovation, France.
- **Process Wise** is a commercial product which supports the enactment of process programs written in PML (Process Modeling Language). It is an offspring of the support environment developed by the IPSE 2.5 project under ESPRIT and Alvey sponsorships. It is marketed by ICL in the UK.

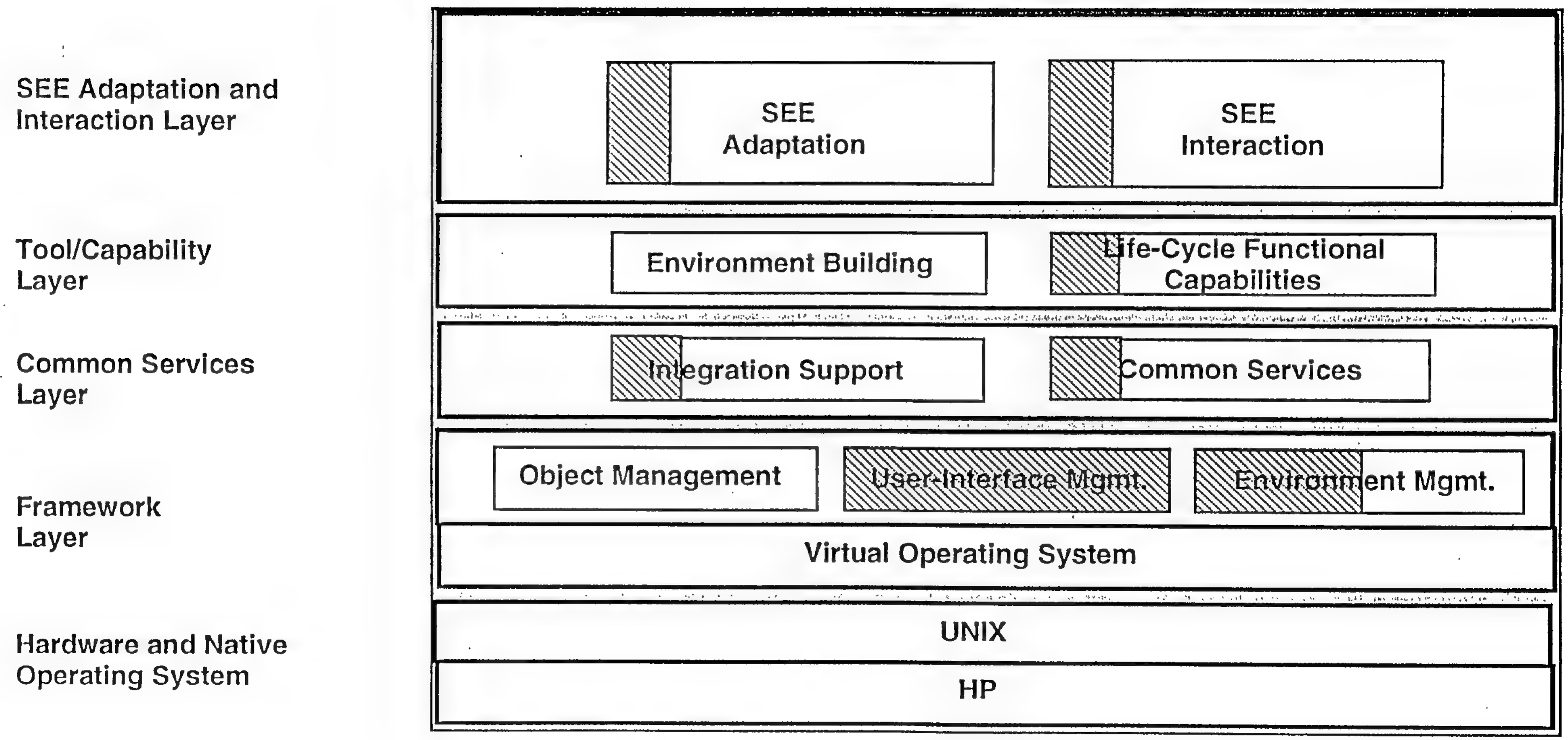
PCTE Mapping to CEARM



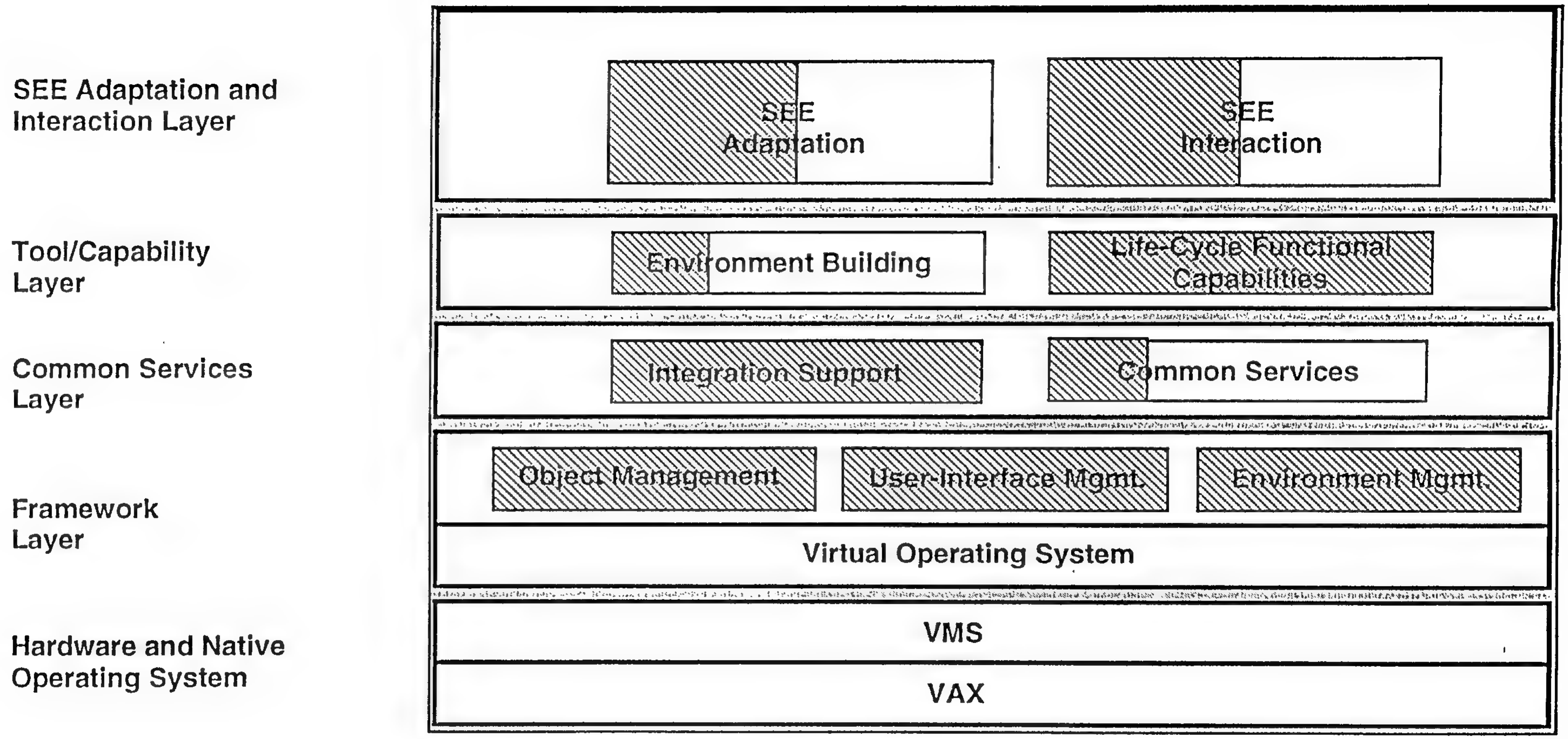
PACT Mapping to CEARM



HP Softbench Mapping to CEARM



SLCSE Mapping to CEARM



SEE Architectural Characterizations

Question: How do we characterize different architectural approaches?

- Virtual Machine (VM) Architectures
- Data-centered Architectures
- Object-oriented Architectures
- Client-server Architectures
- Broadcast Architectures
- Bus-based Architectures
- Control-centric Architectures
- Life-cycle Process-based Architectures

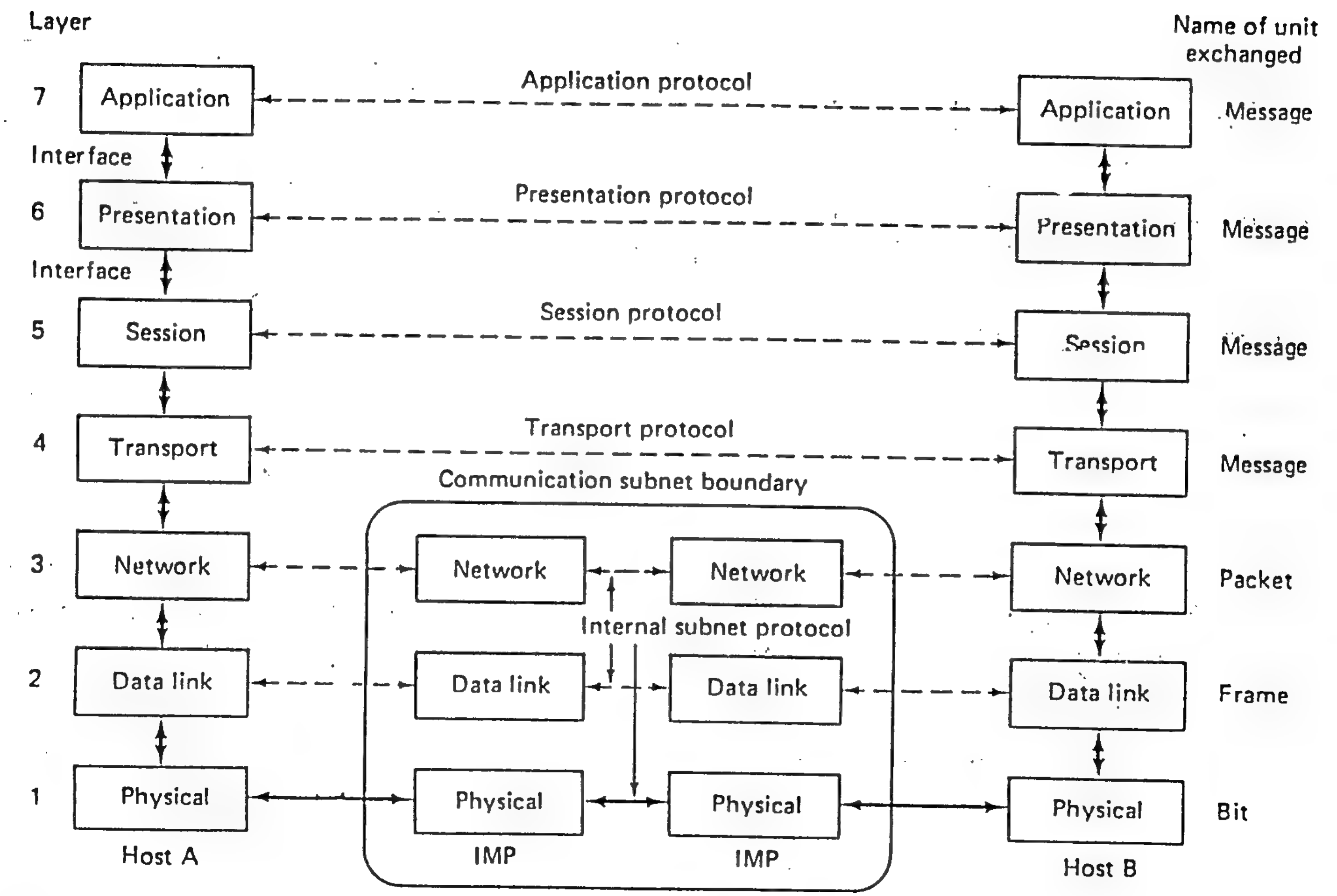
SEE Architectural Characterizations

- **Virtual Machine (VM) Architectures.** The components of an environment are organized into layers of implementation support with lower layers supporting the implementation of higher ones.

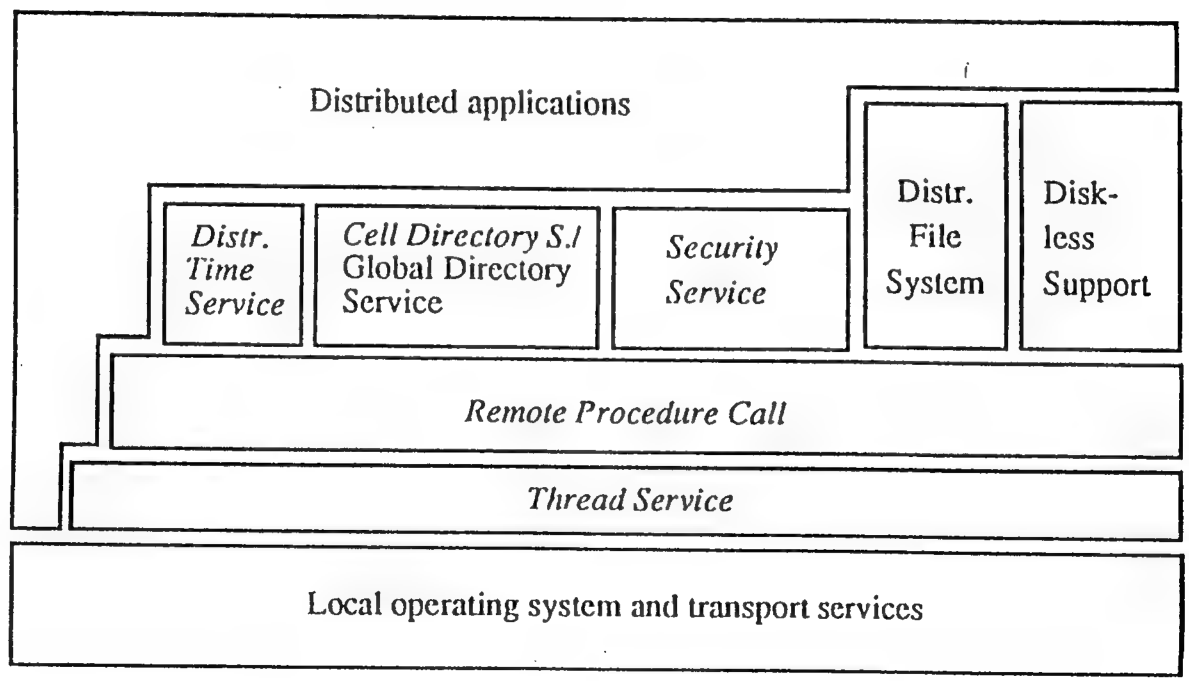
  Comments/Issues:

  - Typically associated with multiple levels of abstraction and/or implementation
  - May isolate functionality and services with well defined interfaces (APIs)
  - Support Open System Environment (OSE) - a conceptual layered framework providing a context for user requirements and standards specifications.
  - Support portability (by making applications independent of the lowest components (e.g., OS)
  - Support standards (e.g., POSIX) and middleware use (e.g., DCE)
  - **Is this used as an architecture or an architecture model?**

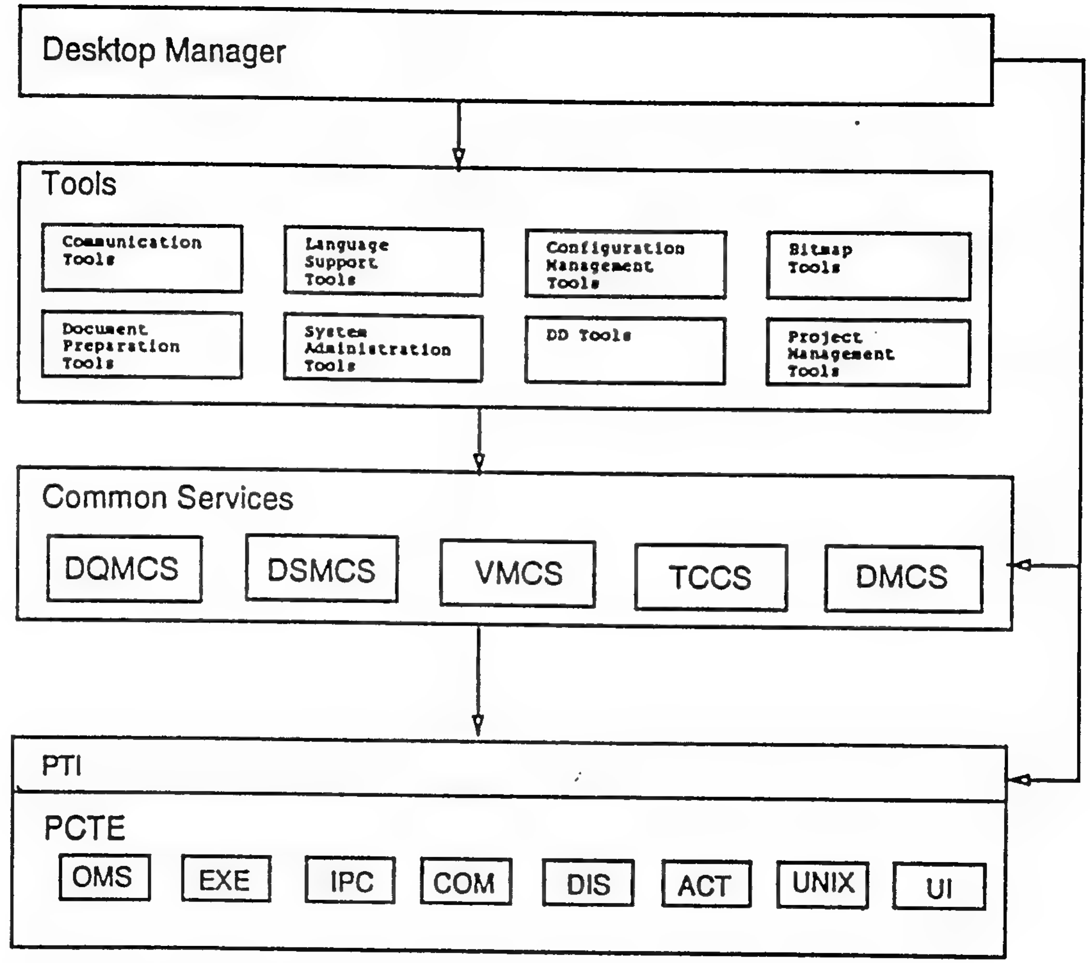
ISO Open Reference Model



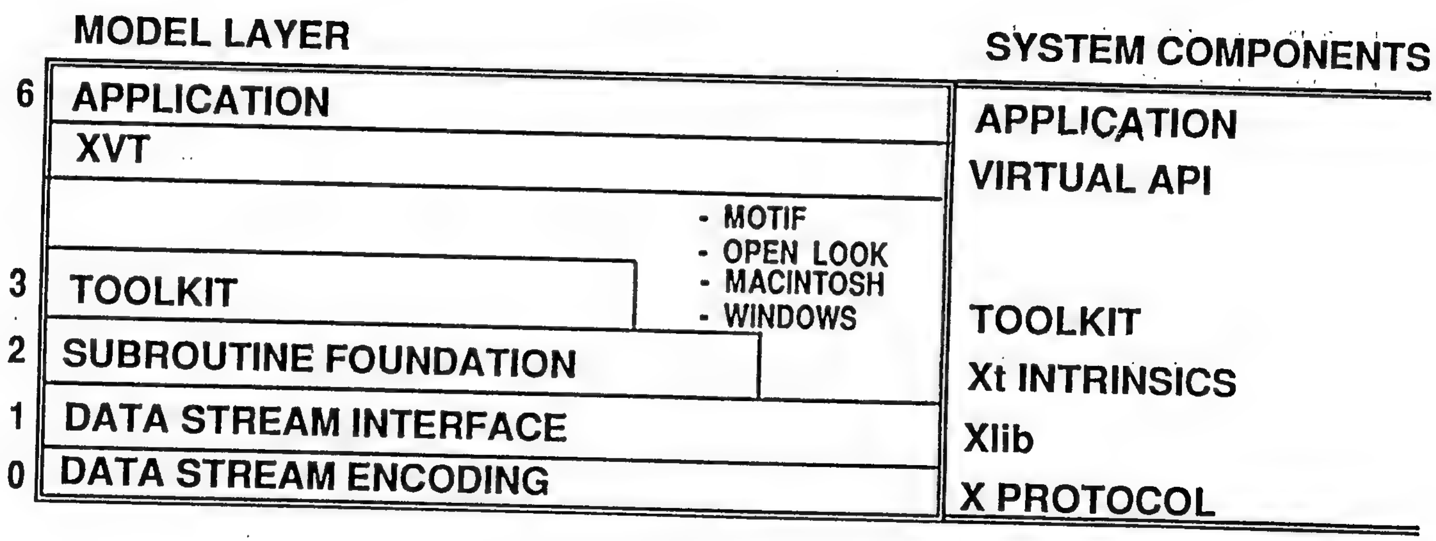
DCE picture


Distributed applications
Distr. Time Service
Cell Directory S./ Global Directory Service
Security Service
Distr. File System
Disk-less Support
Remote Procedure Call
Thread Service
Local operating system and transport services

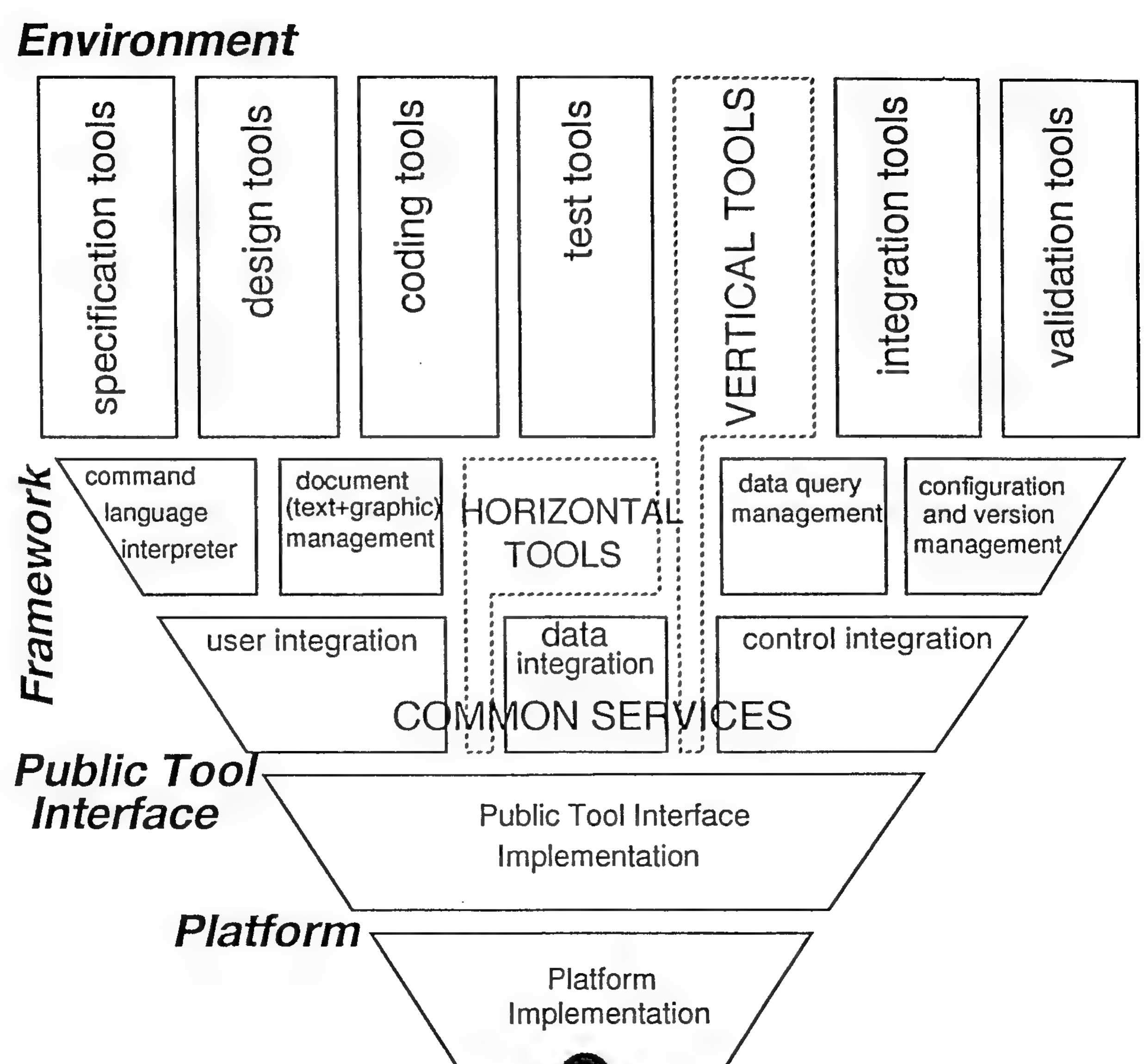
PACT Architecture



X-window layered model

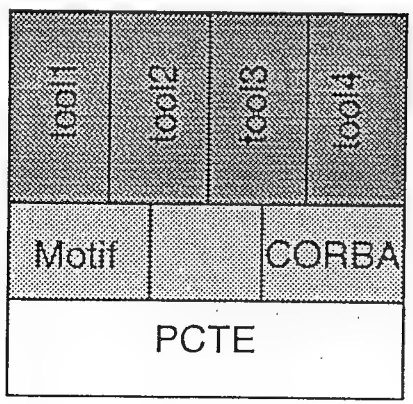


PCIS SEE model

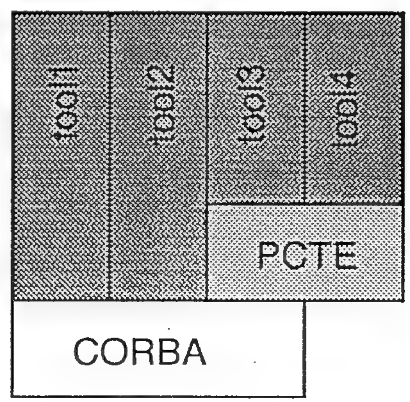


CORBA and PCTE: Architecture Views [by Bremeau]

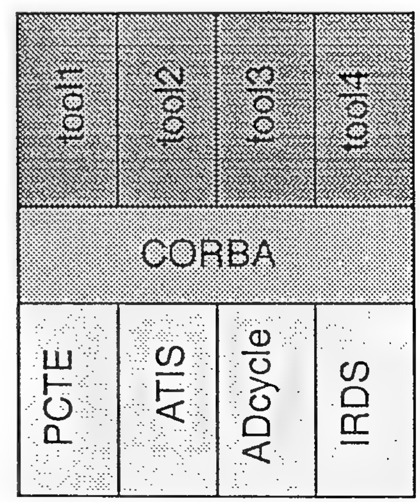
CORBA as an integration service on top of PCTE to cover the control integration area.



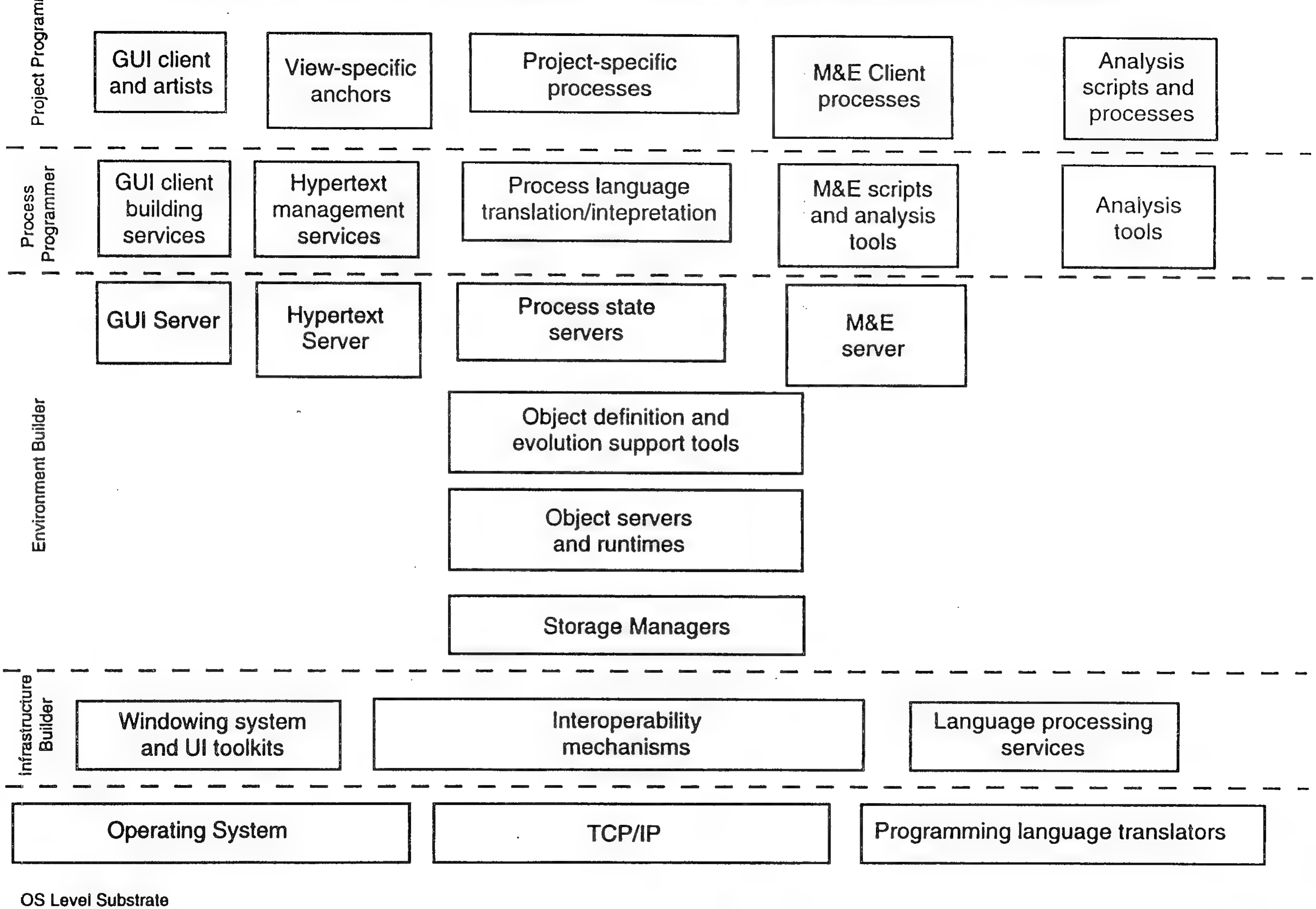
CORBA as the heart of the architecture and by the way a means to implement PCTE.



CORBA as a "software bus" to connect and integrate different tools with different repositories.



Typical Arcadia-1 Components and Virtual Machine Layers



13 February 1994 v2

SEE Architectural Characterizations

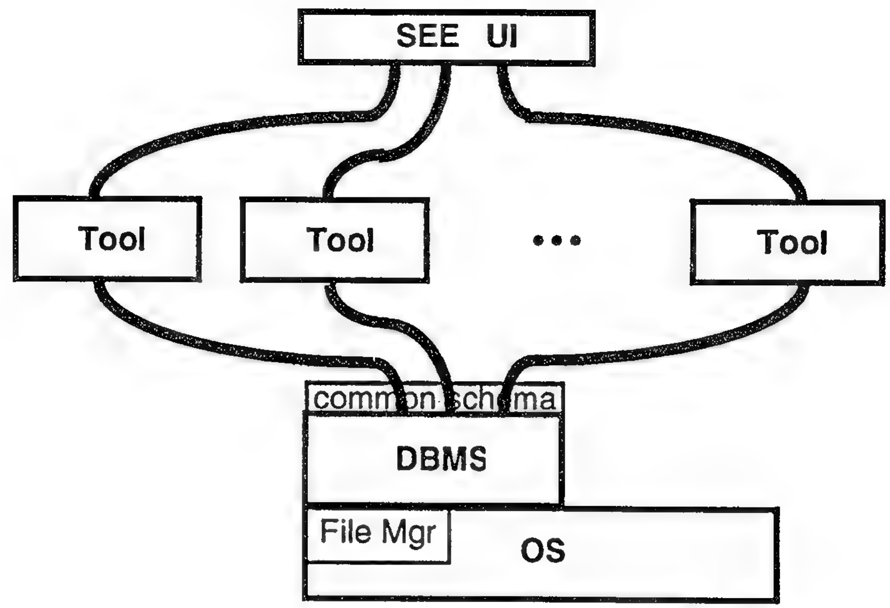
- **Data-centered Architectures.** The components of an environment are organized with the DBMS (or information repository) at the core and tool interoperability is via the database.

  Comments/Issues:

  - Communication between tools and DBMS can be via message passing.
  - Logical and physical centralization versus

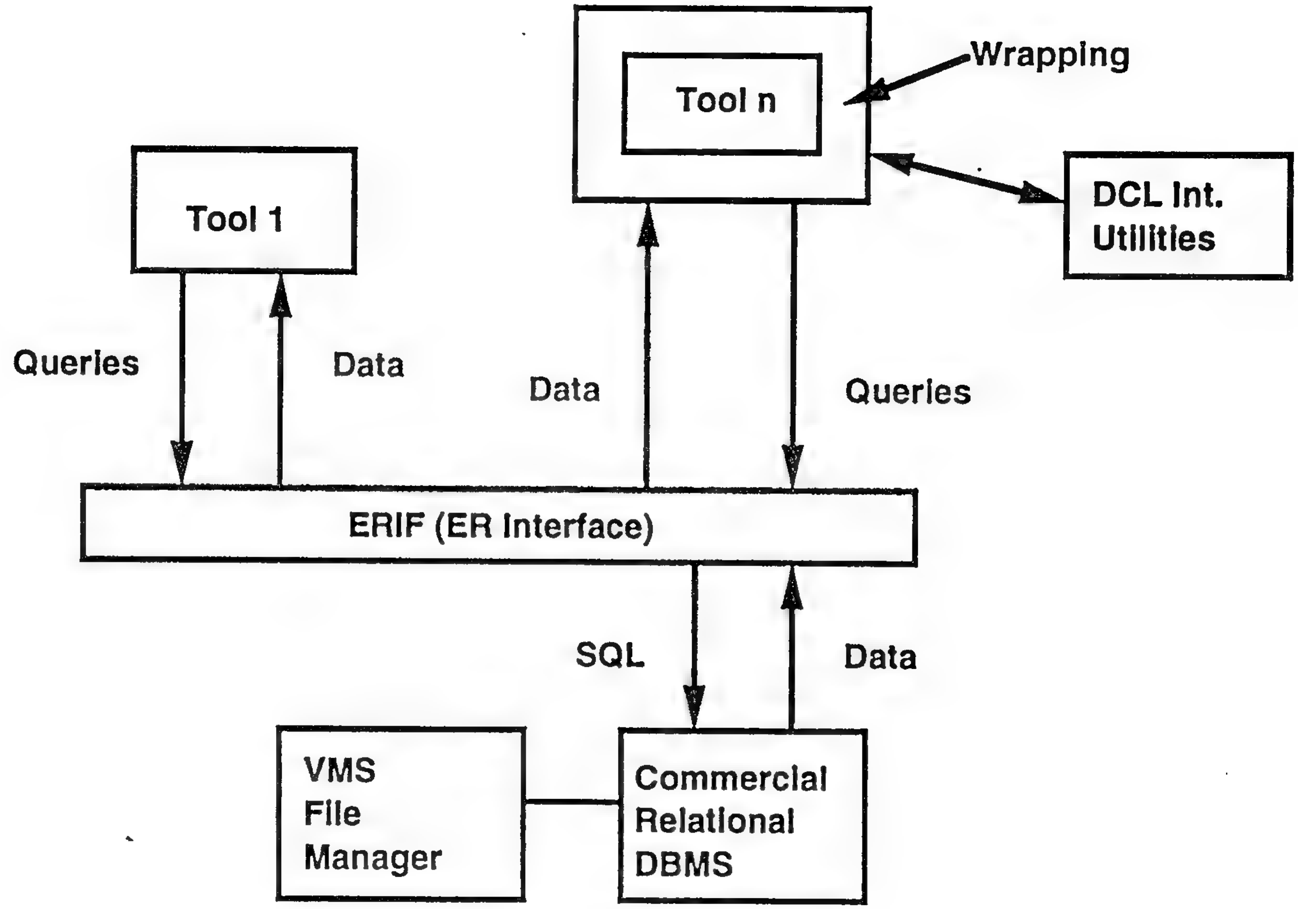
    logical centralization and physical distribution vs ...

SEE Architecture Evolution (cont.)

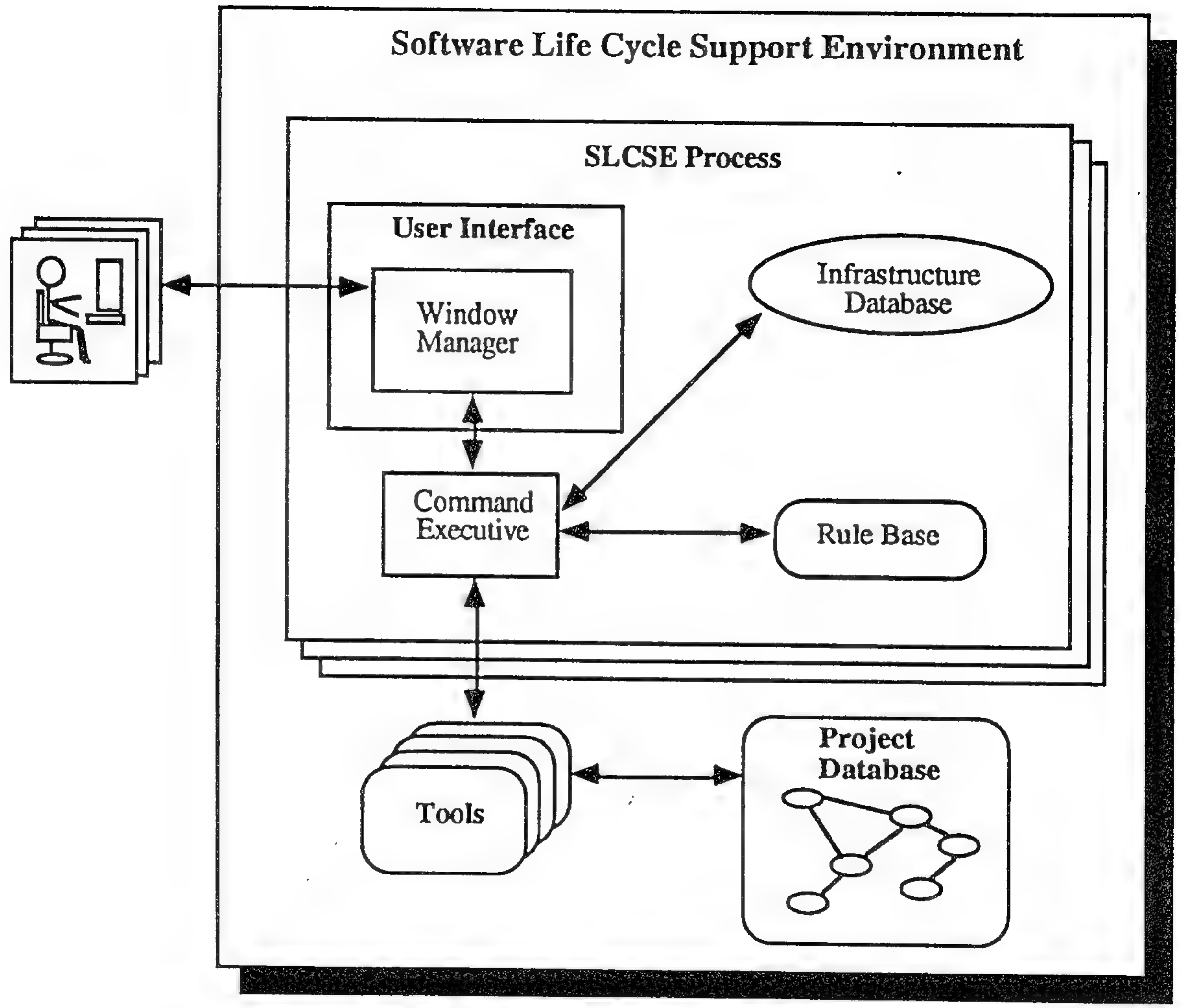


- tools support life-cycle activities
- data is stored in database manager (fine grain)
- common schema provides common semantics

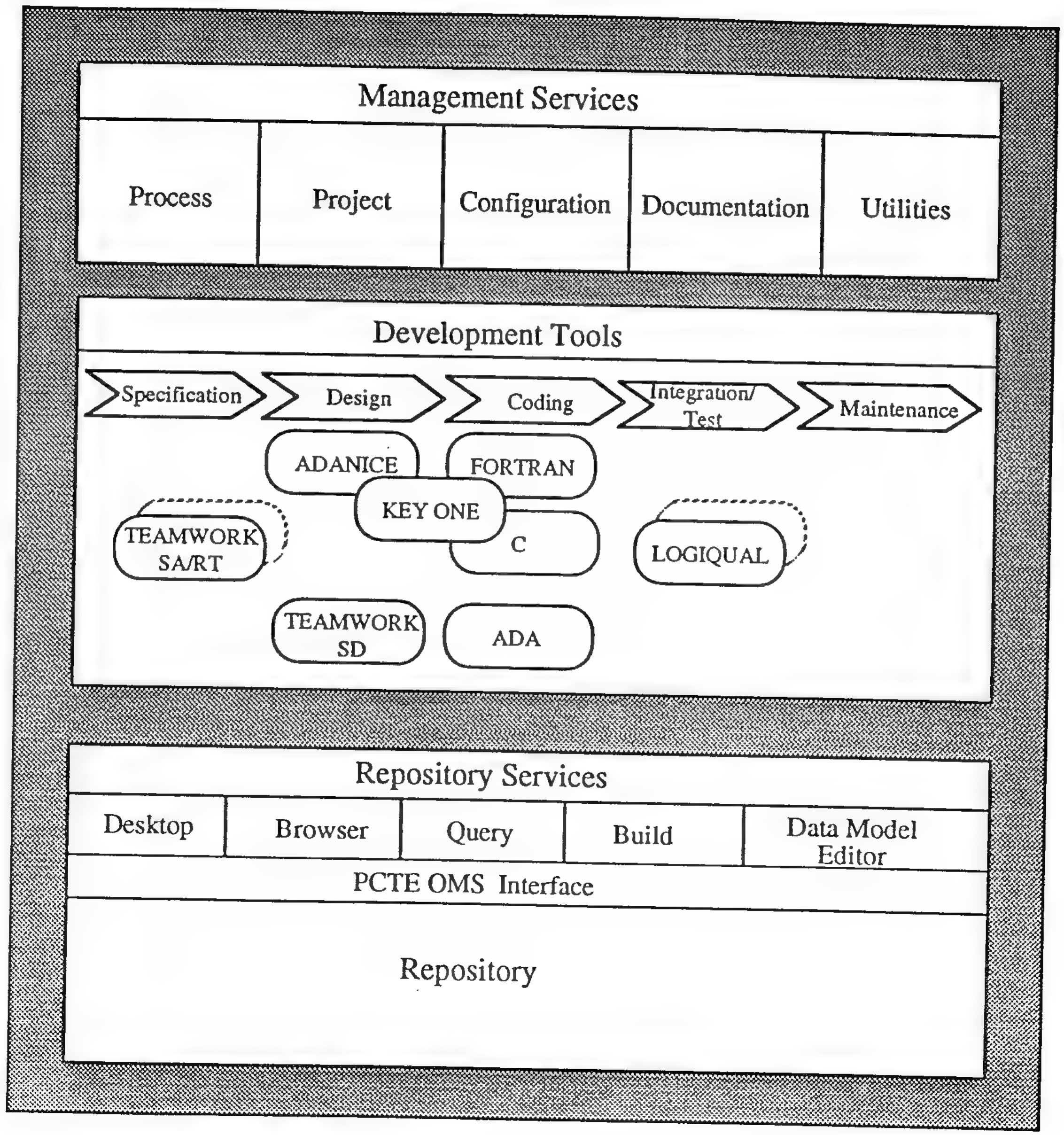
SLCSE Database Implementation



SLCSE Process Architecture (simplified)

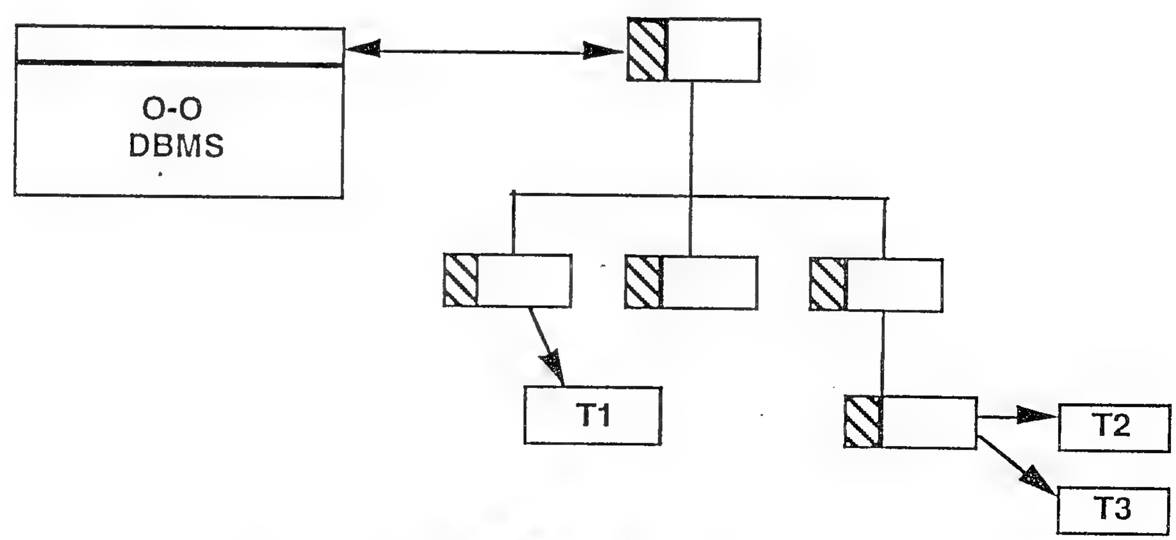


EAST Architecture



- **Object Oriented Architectures.** The environment is organized in a way where the tool components are tightly associated with (or are part of) the DBMS (e.g., information repository); or the components are organized as objects which are accessed via operations.

  Comments/Issues: What are the implications of having the tools implemented as methods, only accessed via the methods, or a combination of both?



- **Client-Server Architectures.** Client and server processes, typically residing on different hosts (possibly remote from each other) communicate via RPC or messages. When clients can provide services to each other, communication is peer-to-peer.

  Examples: database servers, file systems, clocks, distributed systems.

  Most RDBMS vendors offer their client/server tools: Oracle SQL Forms, Ingres 4GL, Windows 4GL, Informix 4GL and Sybase APT Workbench.

FIG. 3 The Structure of Object Request Broker Interfaces

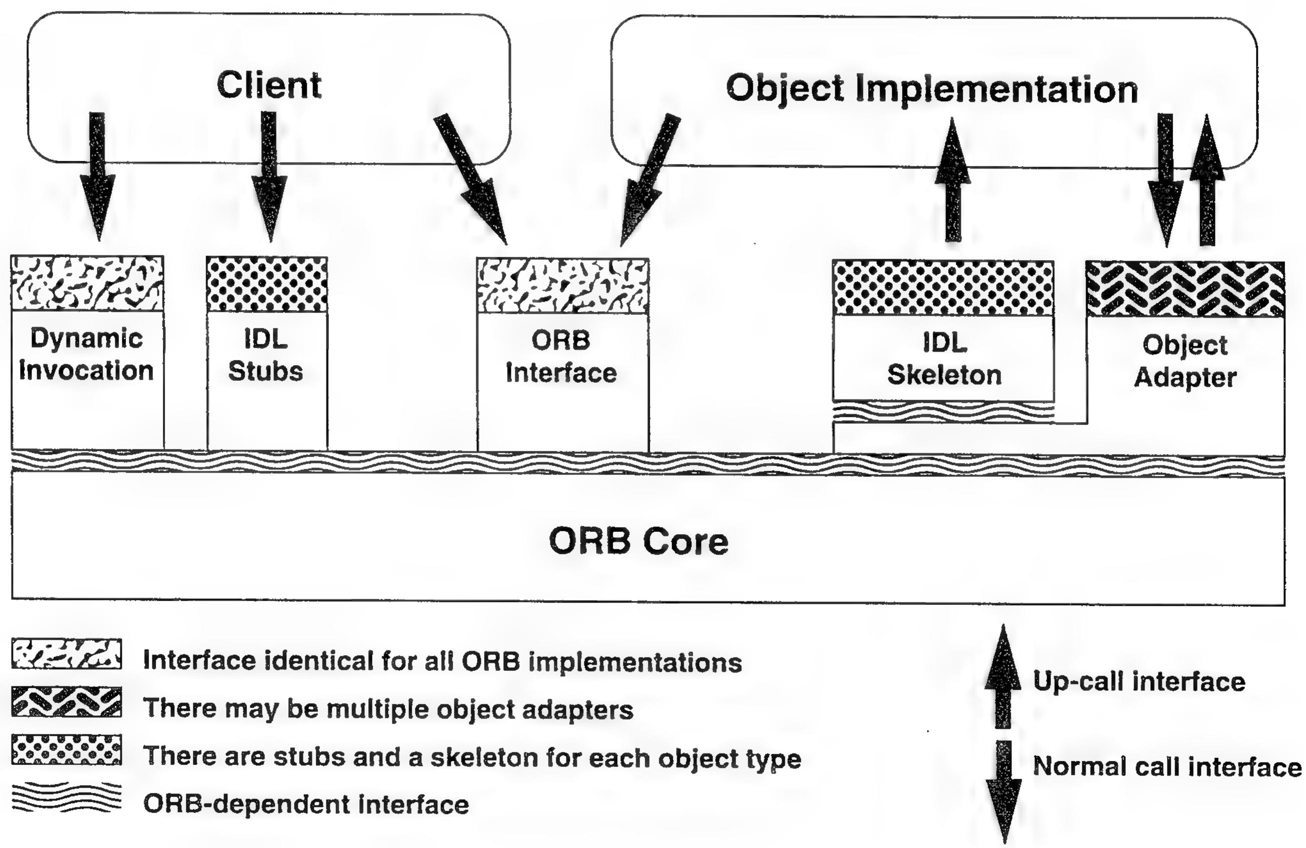


FIG. 3 on page 29 shows the structure of an individual Object Request Broker (ORB). The interfaces to the ORB are shown by striped boxes, and the arrows indicate whether the ORB is called or performs an up-call across the interface.

SEE Architectural Characterizations

- **Broadcast (Message passing) Architectures.** The environment is organized with the components as a network of processes that typically interact through message passing or broadcasting.

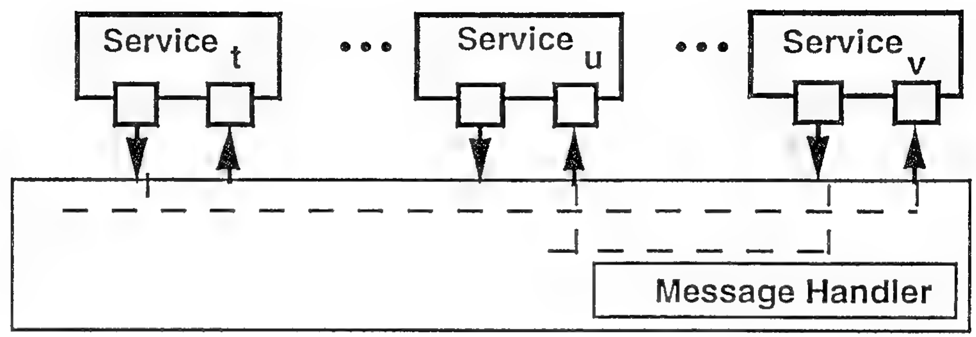
  Comments/Issues:

  - Does the message passing server take care of communication only or is also in charge of data translation and/or constraint management?
  - Typically tools are wrapped for reuse

- **Bus-based Architectures.** (sort of like broadcast-based) The environment is organized with the components interacting via a common communication medium called Software Bus, which takes care of data translation and communication. processes that typically interact through message passing or broadcasting.

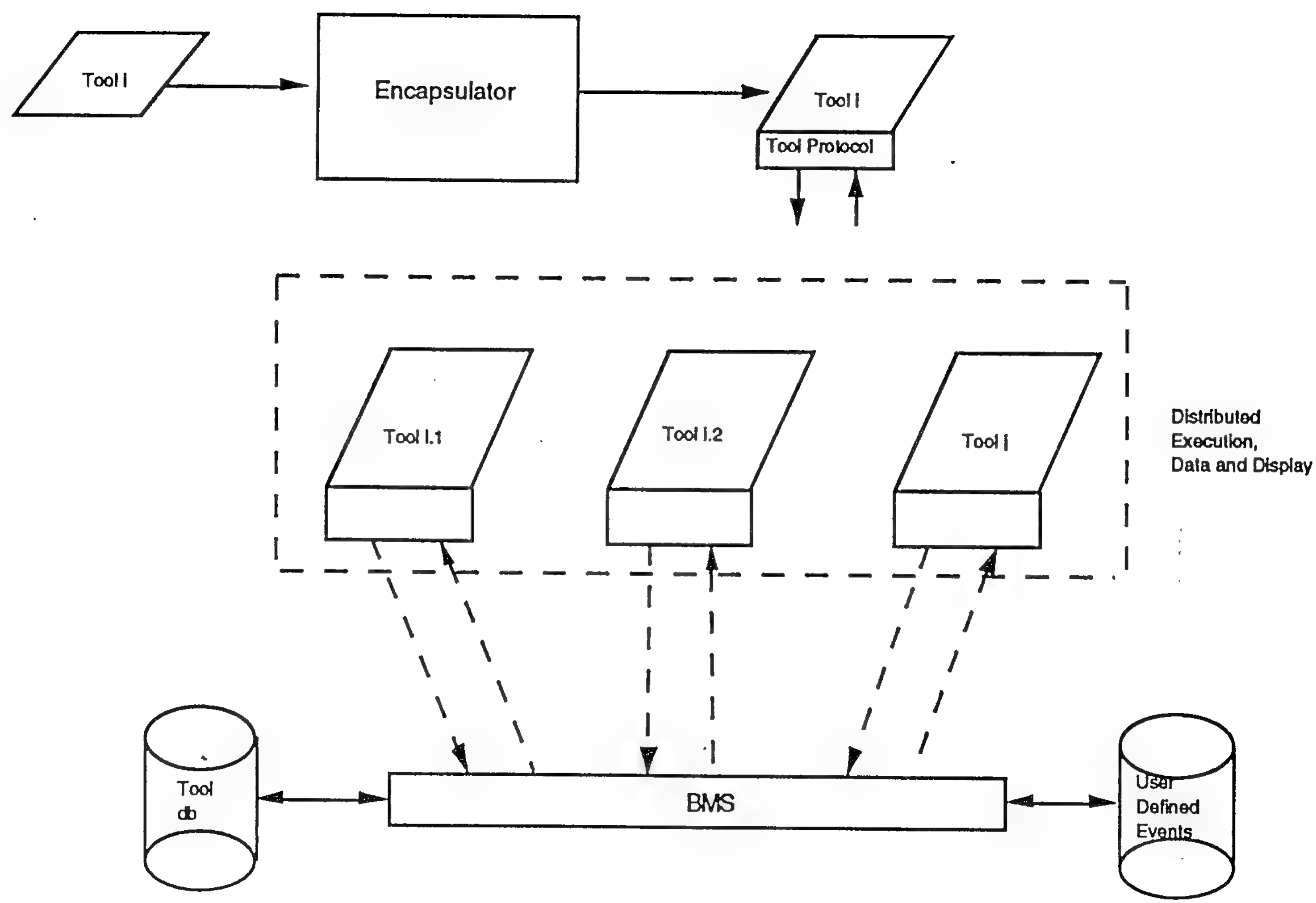
  Comments/Issues:

SEE Architecture Evolution (cont.)

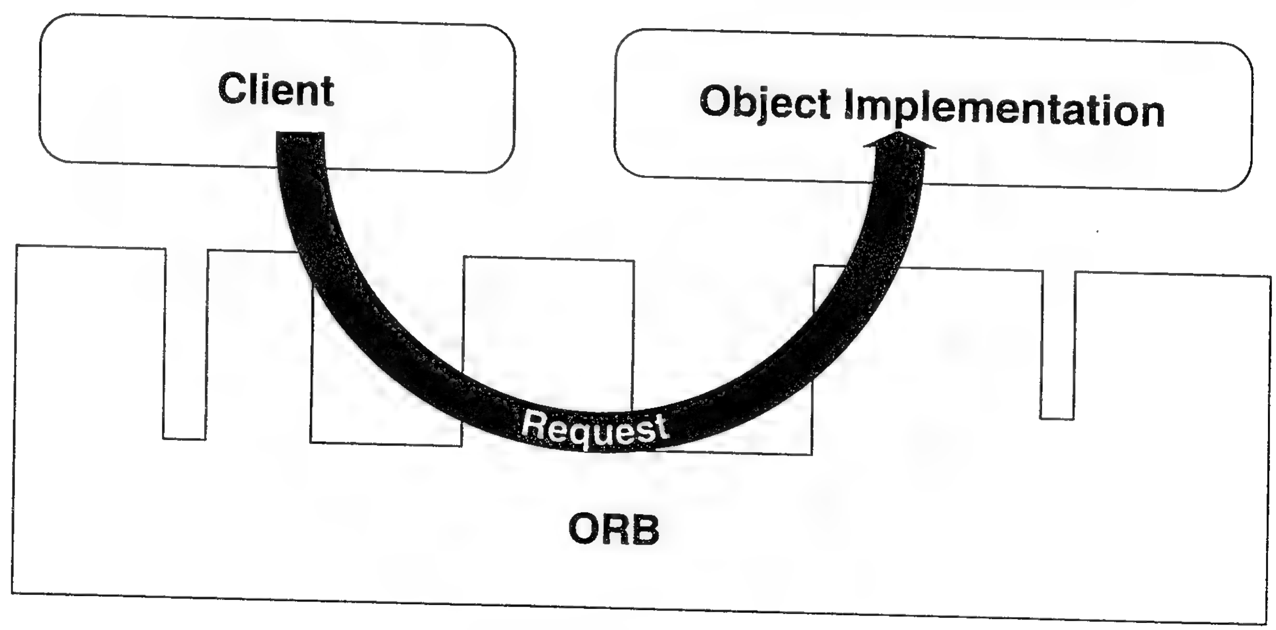


- tool communication via messages, e.g., HP's BMS, SUN's tooltalk, DEC's Fuse , CORBA
- event driven communication
- wrapping tools for reuse

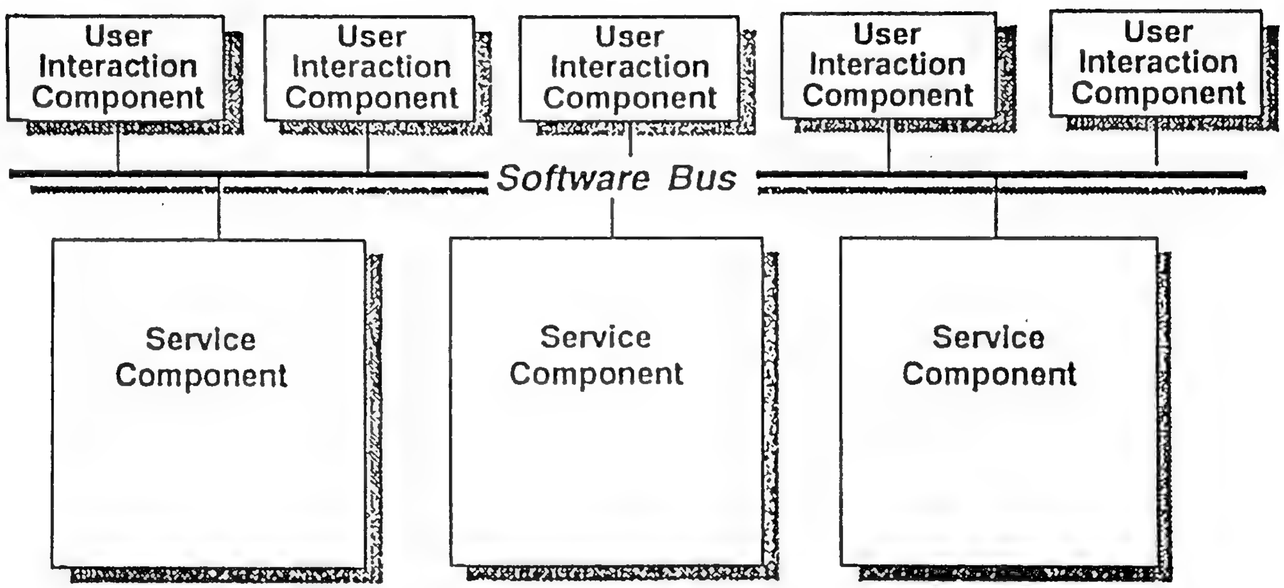
HP Softbench Architecture



Object Request Broker (ORB)



ESF Software BUS Architecture



Software Bus Comparison Framework (Penedo - work in progress)

- Communication Model
- Run-time Behavior
- Data Granularity and Type
- Multilingual Support
- Data Translation
- Protocols
- I/O Synchronization
- Triggering
- Threads
- Scope
- Distribution
- Component Interface Specification
- Registration
- Exception handling
- Security, ...

SEE Architectural Characterizations

TRW

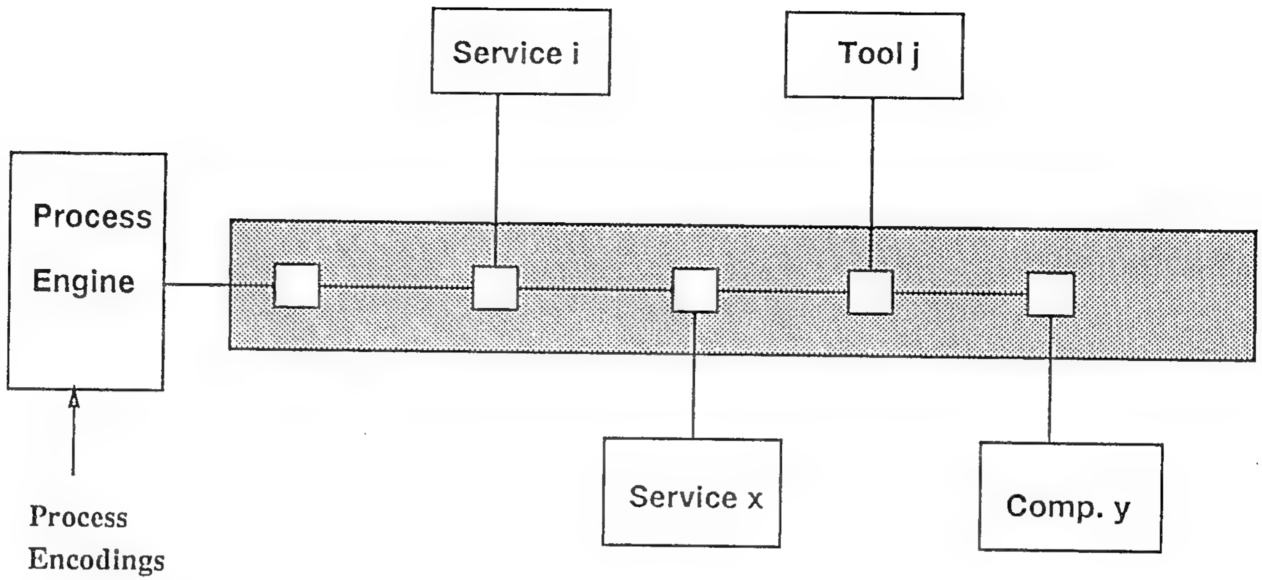
- **Control-centric Architectures.** The environment is organized with its supervisory system at the core and its components are organized in terms of the flow of control among them.

  Examples: expert systems, possibly reactive database systems

- **Life-cycle Process-based Architectures.** The environment is organized in a way where its life-cycle process engine is at the core and its components are organized in terms of the flow of control prescribed by the process.

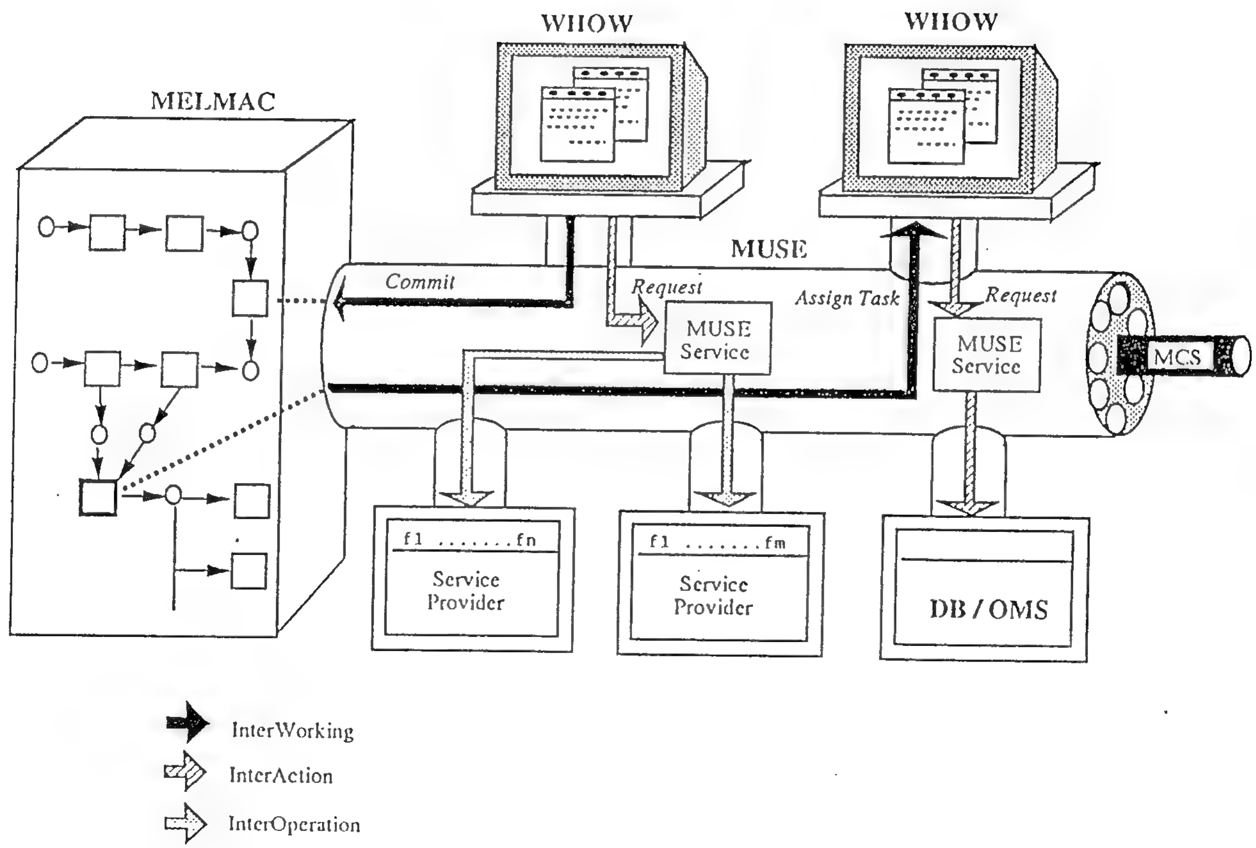
  Examples: Kernel 2/R, Marvel, Leu

Process Centered Architectures

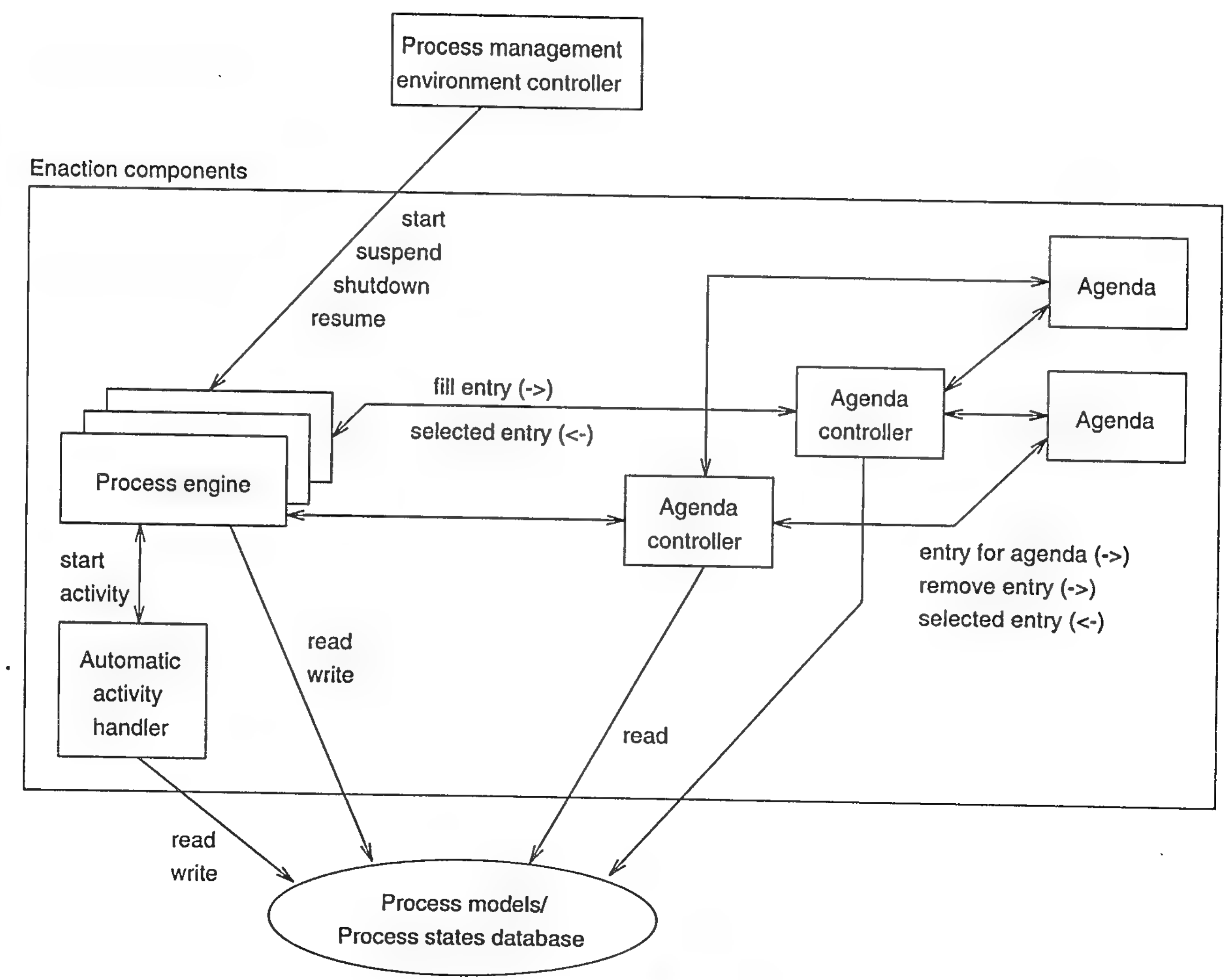


- Environment Execution driven by process, e.g., Marvel, Kernel 2/R, Arcadia
- Process Engine executes process and possibly controls tool sequencing
- Process Engine vs Environment Manager functions will vary

Kernel/2R Architecture

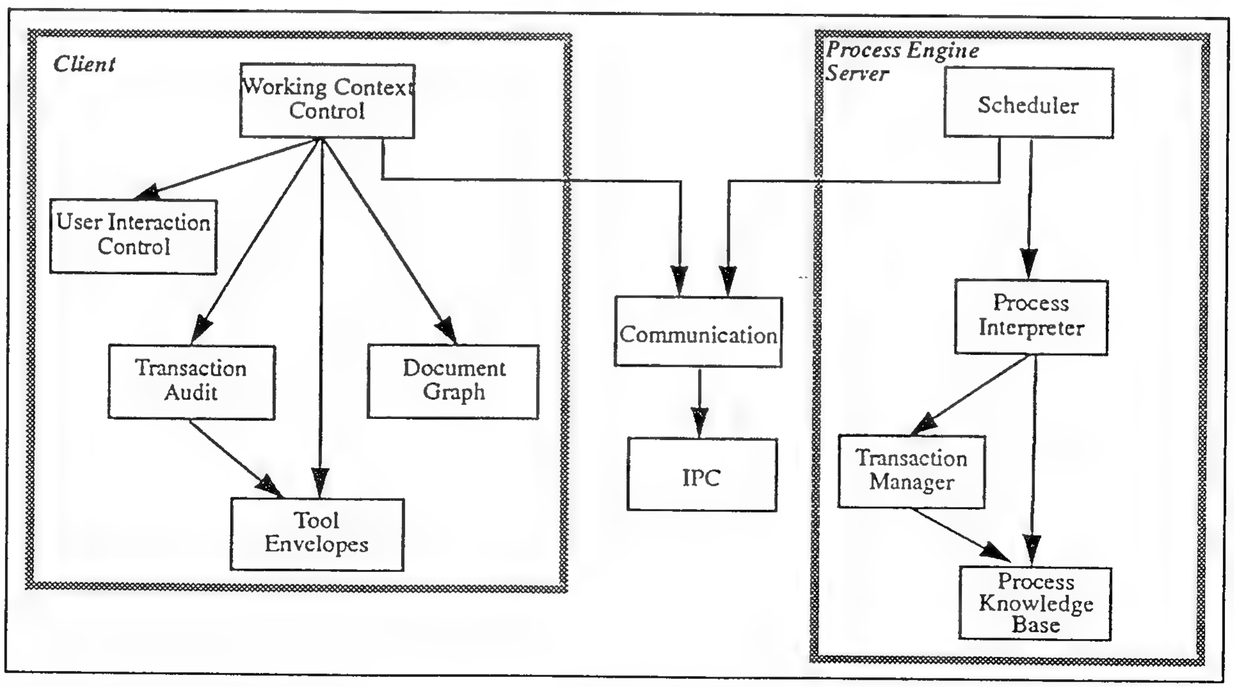


LEU System Architecture



Merlin Architecture

The architecture of Merlin implementing the above sketched transaction mechanism is based on a client server model with the process engine as the server and several clients managing the working contexts. Figure 1 represents an overview on the Merlin architecture. Transactions are initiated by the user via the user interaction control component or by the process interpreter which also initiates transactions to access documents by automated activities like compile.



Architecture Patterns Garlan/Shaw

- **Pipes and Filters.** Each component has a set of inputs and outputs. Output begins before input is consumed.
- **Data Abstraction and object-oriented.** Components invoke functions and procedures from each other. Implementations hidden from clients.
- **Event-based, implicit invocation** Components broadcast events to which others respond.
- **Layered Systems.** Hierarchical support to layers above.
- **Repositories.** Central data structure and independent operators.
- **Table driven interpreters.** A virtual machine is produced in software.

SEEs: Models, Characterizations and Examples
Outline

- Concepts
- Reference Models
- SEE Architectures and Examples
- **Integration**
- Lessons Learned

Integration vs Architecture

- Many times architecture and integration are treated synonymously
- They mean different things; integration is one ingredient of architectures
- Flavors of integration
  - Kinds of integration
  - Degrees of integration
  - Techniques in support of integration
  - Mechanisms in support of integration

SEE Integration Characterizations

- Wasserman's 5 types: platform, presentation, data, control, process
- Other related definitions:
  - Rudmik's framework factors: information, control, user interface, method
  - Brown and Bc Dermid views: (user) interface, technical, tool, team, management
  - ESF five layers: machine, object, process, method-integration, method-uniform
- Penedo's perspectives: architectural, user interaction
- Thomas and Nejmeh properties: associated with relationships between environment components

More work needs to be done in this area.

Integration: commonly used term

[appears in Wasserman 89]

- **Platform integration.** The set of system services that provides network and operating systems transparency to applications
- **Presentation integration.** Tools that share a common "look and feel" from the user's perspective
- **Data integration.** Support for data sharing across tools.
- **Control integration.** Support for event notification among tools and the ability to activate tools under program control
- **Process integration.** Support for a well-defined software process.

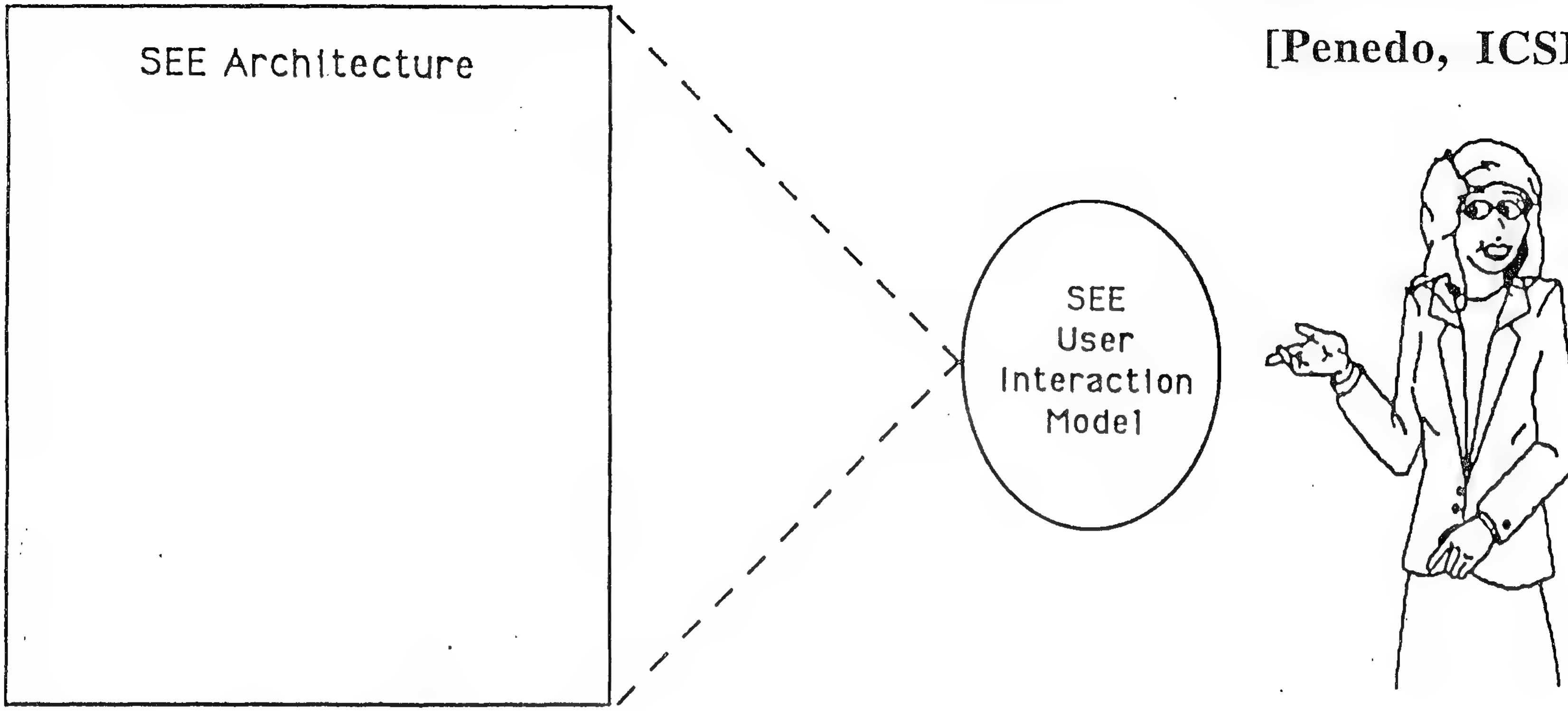
SEE integration perspectives

- **SEE User Perspective.** emphasizes user interaction, i.e., how the user communicates and perceives the environment to operate in order to perform life-cycle activities/tasks.
- **SEE Tool/Environment Builder Perspective.** emphasizes environment organization and architecture, i.e., how the environment is architected and how its various components interoperate.

- Comments:
  - perspectives often confused in literature
  - one does not necessarily imply the other

Integration in SEEs: different perspectives

[Penedo, ICSE13]



Env./Tool Builder Perspective:

- Platform Integration
- Data Integration
- Control Integration
- Communication Integration
- Process Integration
- User Interface Integration
- Tool Building Integration

Env. User Perspective:

- Invocation Integration
- Object Integration
- Tool Kit Integration
- Process Integration
- Presentation Integration

Existing mechanisms/systems supporting integration

- Platform Integration: POSIX, Mach, POSIX.2, DCE
- Data:
  - Repositories/DBMS: Oracle, Sybase, Montage
  - Repository interface: PCTE, ATIS
  - Metadata: PCTE, IRDS
  - Database Languages: SQL, ODBMS
  - Interchange Formats: CDIF, EDIF, PDES, PHIGS, GKS, DIF, ASIS, Postscript, ASCII, ODA/ODIF, CGM, SGML, STEP, HTML MAPI
- Control:
  - Messaging Mechanism: CORBA, OLE, BMS, ToolTalk, X3H6 Message Architecture
  - Interface Description Language: IDL (CORBA)
  - Message Semantics: X3H6 Messaging Servicegrams draft

Existing mechanisms/systems supporting integration (Cont.)

- Presentation:
  - Presentation Mechanism: X-Windows, CDE
  - Presentation Format: MOTIF, CDE
- Process:
  - Process Definition Tools: IDEF, other Design Tools
  - Process Execution Tools: Process Weaver (CAP Gemini), Process Wiser (ICL), Synervision (HP)

PSEE concepts/needs:

- need for better PSEE characterizations
- support individuals, teams, project and organization
- support social, organizational and task networks
- allow integration of technical and managerial data
- support data sharing and process consistency
- support different levels of guidance
- wrt enforcement:
  - enforce the process without limiting individual freedom and ability
  - prescriptive guidance is preferable from a psychologyical point of view
  - provide process articulation mechanisms for tracking and assimilating excursions outside the process
- allow proscription as applied to tools
- support distinction between process models used for project user's consumption and their implementation
- process tailors should be unaware of implementation and should concentrate on process models which resemble their world perceptions.

Integration:

- association of tools and process programs should be dynamic
- common interfaces and protocols are integration mechanisms
- tool to process integration will require agreements on: communication, coordination/synchronization, data sharing, presentation consistency.
- We need to distinguish integration from at least two distinct perspectives: architectural and user interaction.
- We need to meta-integrate, i.e., integrate the integrating mechanisms/components.
- Integration needs to be addressed at different levels, e.g., mechanism (i.e., implementation), conceptual (abstraction of functionality) and process (the process to be executed).

SEEs: Models, Characterizations and Examples
Outline

- Concepts
- Reference Models
- SEE Architectures and Examples
- Integration
- **Lessons Learned**

TRW Survey - Summary

- There is little or no agreement on ways to describe systems or architectures.
- Even though the CEARM was not defined in full, it was useful as a common framework for describing, discussing and comparing distinct efforts. Graphical depictions, for example, describe at a glance the focus of the existing systems.
- Performing the mappings increased our understanding of the existing systems.
- Performing the mappings identified the least known areas, Environment Management and Integration Support. Object Management is the best known area.
- Mapping to specific services is needed, in order to capture the "meat" of the systems and to provide for meaningful comparisons.
- There is a need to distinguish between services "provided directly" by a system/effort from the services which are "supported" (i.e., not provided directly but can be emulated using existing services/capabilities).

Lessons Learned and Observations
- per RM grouping -

Virtual Operating System.

- Few systems provided support for this grouping of services; worrisome since portability of components is a key requirement for industries where the platforms for SEEs vary. Movement towards PCTE adoption supports this need.

User Interface (UI) Management.

- X is the de-facto standard platform for UNIX-based UI management and is used by most UNIX-based environments. Higher level UI toolkits (e.g., Motif, Openlook) are also widely reused.
- Many UI development systems are starting to appear, providing higher level and generative capabilities.

Lessons Learned and Observations - per RM grouping -

Object Management.

- Most explored to date, most services provided, at different degrees of functionality (e.g., data models, versioning model) - more rigorous means are needed for comparison
- A few needed services not provided are being researched (e.g., long transactions, common models, dynamic schema change)
- Many use the common repository approach for achieving data integration. A few support the multiple heterogeneous data store approach.
- Common repositories are not enough; commons schemas need also to be adopted. A variety of approaches is being used including views, encapsulators, and translators, to integrate tools to repositories.

Lessons Learned and Observations - per RM grouping -

- Providing functional view of O-O approaches was very insightful on how the objects interact and relate to each other (beyond inheritance). For example, the ATIS object oriented model also includes services for integration support, common services and environment management.
- The RM provided insufficient means to describe repositories which are local to a cluster of tools. This distinction is important to be described for existing systems since it reflects important architectural properties.

Environment Management.

- The Environment Management (EM) grouping is the least explored/agreed grouping in the RM and implementation worlds, especially with the current advances in PSEEs. Many complex services are being explored supporting distribution, event management, measurement, dynamic modification, etc.

Lessons Learned and Observations
- per RM grouping -

- EM services in environments take different forms:
  - ESF and Arcadia are based on process programming and event-driven paradigms; Kernel/2r provides environment management at the user and project level. Process servers and event managers provide services for environment management.
  - Atherton provides an object-oriented database and uses the functionality attached to objects as the basis of Environment Management services.
  - CAIS and PCTE both have specific objects which are used to control active entities. In this case the OM may support environment management.
  - HP Softbench and K/2r provide an integrated communication and execution approach based on message broadcasting.

Lessons Learned and Observations - per RM grouping -

- SLCSE provides rule-based, user role oriented run-time environment management.
- EAST provides a task- or process-oriented approach.

Integration Support.

- Addressed from a wide spectrum of perspectives and degrees. Although the tool integration approach employed by most environments surveyed sometimes appear similar, the implementation mechanisms differ widely. For example:
  - PCTE addresses one aspect of data integration, the repository interface; PACT adds common schemas; and SLCSE adds componentry in support of data translation and filter generation.
  - Softbench supports presentation integration by encapsulating tools and Arcadia provides a full fledged UI development system.

Lessons Learned and Observations - per RM grouping -

  - Softbench provides a broadcast message control mechanism; Arcadia provides point-to-point client-server control support; and Kernel/2r provides a communication-mediated bus with generative technology.

- Software bus technology is emerging as a strong mechanism in support of control (and possibly data) integration, as exemplified by Arcadia's Q and K/2r's Muse systems.

- The task of integrating tools is still non-trivial and much work is currently under way investigating better characterizations and better mechanisms towards this end. The CASE community and government agencies have ongoing efforts: X3H6, CASE Interoperability Alliance and CASE Communique.

Lessons Learned and Observations - General -

- *Usefulness of service-based RMs.*
  - Useful as a common framework for describing, discussing and comparing distinct systems, especially helping in determining: which components to compare across systems; areas of overlap; and areas not provided.
  - Useful to describe the functions of a SEE and decouple service from actual componentry.
  - Useful as a vehicle for understanding SEE componentization issues.
  - Mappers felt the mapping exercise helped them understand their own architecture better, and to determine places where designs or documentation could be improved.
  - Describing services from different dimensions (e.g., conceptual, external, operations, rules, metadata, data, types, instances, internal, related services, and examples) does increase understanding about capabilities.

Lessons Learned and Observations
- General -

- *Graphical depictions.* Graphical depictions of RMs are useful but they are not sufficient without accompanying descriptions; at times, they can be misleading.

- *Structure of RMs.* Groupings are not necessarily disjoint and there are areas which cut across all groupings (e.g., integration, security). More work needs to be done on those intersections.

- *Separating SEE user interaction from architecture.* It is mandatory to distinguish between the SEE User Interaction Model (i.e., how the project user interacts with and perceives the environment to operate in order to perform life-cycle activities/tasks) and the SEE Architecture (i.e., how the environment is organized and how its various components interoperate in support of life-cycle activities/tasks). The first view is important to the (project) users of the environment and the second one is important to the SEE builders or adaptors.

Lessons Learned and Observations
- General -

- *Object-oriented systems.* We found that mapping object-oriented frameworks (e.g., ATIS) to the CEARM was somewhat awkward due to the inherent functional grouping of the model and the fact that OO services are embedded in the type hierarchy and in the underlying implementation of the OO framework. However, the mapping did add value to the OO hierarchy by identifying functional clusters of related types and related operations.

- *Integration services.*
  - It is currently very difficult to determine the level of integration of existing environments. The existing integration models are lacking.
  - The set of integration services in RMs is still incomplete and under investigation.

Lessons Learned and Observations
- General -

- Examples of control integration services are: component composition services, communication services (to invoke, suspend, terminate events, requests, etc.) within and across platforms, registration of functionalities (static and dynamic), access control to components, and resource constraint management.

- Examples of data integration services include: data translation, multi-lingual type interoperability, common repository, common query, data interoperability (across repositories, schemas, platforms, etc.), data exchange, internal forms (fine grained common schema), unifying schemas, and data integration policies.

• *Weaknesses of Service-based approaches.* They fall short in supporting the description of: interconnections or dependencies between components, clusters of semantically related components, distinguishing between static and dynamic dependencies, properties of architectures.

Lessons Learned and Observations
- General -

- *Areas requiring further work.*
  - Environment Management and Integration Support.
  - Process management services are preliminary and under evaluation
  - SEE Adaptation grouping needs a better characterization of services; could also be represented as another dimension of the model.
  - Other issues currently under discussion in the community imply areas for further work. They include: how to represent the orthogonal models (e.g., integration), how to use the RM to identify interface areas, enhancements to support architectural needs, and how to incorporate properties (i.e., extensibility).

- Componentry:

  7. repository (single vs multiple, central vs distributed)
  8. "common" data (process??) model
  9. common schema in support of tool data integration
  10. (OS) processing (centralized vs distributed, client-server,federated)
  11. PSEE management and monitoring (component management, measurement, component monitoring)
  12. component communication
  13. process engines (single vs multiple (homo vs heterogeneous), local/central vs distributed)
  14. Tool/component (separation of user interface, separation of functionality, separation of communication aspects)
  15. integration support (data,control,presentation,process)

Architecture Characteristics

- Capabilities/Attributes:

16. concurrent execution and control (multi-user and multi-process)
17. security
18. event handling support
19. fault tolerance support
20. distinction between individual, team, project support
21. SEE execution driven by explicit process definition
22. Dynamic process change (code and data) support
23. interoperability: - persistent data - tool - process
24. exception (error) handling support
25. consistency and inconsistency management
26. granularity of data and control

- User interaction issues:
  27. SEE user interaction paradigm
  28. prescription vs proscription
  29. training/guidance support
  30. decision making support
  31. support user roles
  32. human support
  31. support user roles
  33. process enforcement

Architecture:

- A PSEE where all tools are integrated throught the underlying mechanisms (e.g., OM), provides the facilities needed for Process management. [King]
- Core services needed for process definition and enaction are support for activities and policies (constraints or notification) associated with beginning and end of each activity. [King]
- It is still an open question how much of the required functionality of a PSEE can be provided by tools, that can be integrated in "any" environment, and how much has to be provided by "built in" mechanisms, e.g., active databases, etc. [Dowson]
- A PSEE must support many process engines which are distributed according to the needs of the organization and team. [Holtkamp]
- Initial emphasis should be put in solving problems within a single, homogeneous environment before tackling interoperability issues of heterogeneous environment architectures. [Shu]
- Future PSEE architectures should be federated and supporting the attributes of autonomy (have well defined and controlled interfaces), mobility (provide for a parameterized collection of components which can be collectively moved and function correctly in a new environment) and information malleability (accessibility of data in different forms). [Heimbigner]

- Since PSEEs will exist as loose federated architectures, we need to address the problems of interoperability between partial-environments of varying degrees of openness. [Heimbigner]
- Multi-user PSEEs should provide flexibility in the selection and application of synchronization mechanisms in their support of data and process sharing. Those mechanisms should support: classical atomicity, serializability, long interactive operations and cooperation. [Ben-Shaul et al]

  A mechanism supporting the synchronization of concurrent development tasks has to be very flexible as it has to support different needs and situations. [Wolf]
- Inconsistencies between the enactable process and the actual state and behavior of the process (process performance) are bound to happen. Therefore, there is a need for monitoring and support for corrective actions when inconsistencies like these arise. Specific mechanisms include: event handlers, enactment time exceptions, access to persistent state of proess, transaction mechanisms. [Dowson]
- Storing process state together with data (in a DBMS centered SEE) allows for a considerable degree of correlation to be kept between the process and the products of the process.
- PCTE can be used as an underlying mechanism to support certain styles of process definition, enactment, and maintenance. [Simmonds]
- Static process descriptions tend to be out-of-date before the process is instantiated and put into effect. That is because by the time a process description has been captured, it reflects some previous state. In order to capture "living" process descriptions, we must support process change and embed passive capture tools in PSEEs. [Singh]

- To support the evolution of data and programs written by a large number of developers in a way that has minimal impact on existing, stable parts of the system, the SEE (like an OODB) needs a uniform model of change and a family of mechanisms supporting change in an efficient and minimally intrusive fashion. The Open OODB's computational model and mechanisms might also be appropriate for SEEs. [Wells]

- Separation of process modeling and implementation is a must, and, semi-automated means for translating between process models and their implementations are necessary in order to support tailorability, extensibility, and time-constraint needs. [Penedo]

- The only *essential* component with respect to tool and process support in an environment is its communication media. For database-centered environments, this is, of course, the database. For bus environments, this is the software bus. Nevertheless, particular capabilities (in the form of tools) that are not essential can still have a pervasive influence on the structure of all the other tools in the environment. [Heimbigner]

- In open and federated environments, enforcing any globally mandatory constraints will be difficult, if not impossible. [Lolo's adaptation of an email discussion between Simmonds and Heimbigner]

Outline

Introduction

- Rationale - Cost and Productivity Highlights
- Process - concepts, examples
- CASE/SEEs - concepts

SEEs: Models, Characterizations and Examples

- Concepts
- Reference Models
- SEE architectures types and Examples
- Integration
- Lessons Learned

Conclusions

SUMMARY

- "Modern" SEEs and tools are generally immature
  - "immature" will almost always be the characterization of any new technology, or one advertised to be *"best"* or *"most modern"*
- There are no *cheap* SEE solutions
  - historically, free tools have been worth their price
- The most effective SEE solution is not necessarily the most expensive one
  - *"you get what you pay for"* is not always literally true in comparing values of alternate solutions
- SEE/tool selection is a strategic decision
  - this selection process warrants a coordinated organizational plan
  - and it warrants a planned investment
- ***Be Process-Driven!***
  - select SEE and tools based on project process(es)!
  - SEE/tool selection is itself an important process!

Conclusions -1

Many CASE tools exist for different platforms, interfaces, standards, and protocols.

- becoming much more robust
- seen as essential in managing the large amounts of data generated (designs, code, documentation,....)

Some CASE coalitions and point-to-point solutions are being used on commercial projects.

A few integration technology products are available and are slowly being used in operational projects.

- inter-tool communication (e.g., ToolTalk, BMS)
- common data management (e.g., RDBMS products)

Some support for common CASE standards, but these are not yet commonly in use.

But....

No "Off-the-Shelf" Solution Exists

- useful components are available
- end-users must do a lot of work to make use of them in a consistent, robust fashion

A lot of maneuvering, activity, and promises

- porting tools to new platforms and integration products
- "strategic" alliances between vendors

Care needs to be taken in evaluating vendor claims

- there is no common understanding of
  - what integration means
  - what integration is needed
  - how integration is best provided

Characteristics of Emerging SEEs

- Use of standard graphical user interfaces
- Shared repository - becoming more common
- Access to databases via filters
- Movement towards common models
- Point-to-point tool integration
- Client-server architectures
- Light process support - heavy method effort
- Dynamic interaction among tools
- Tool communication via messages

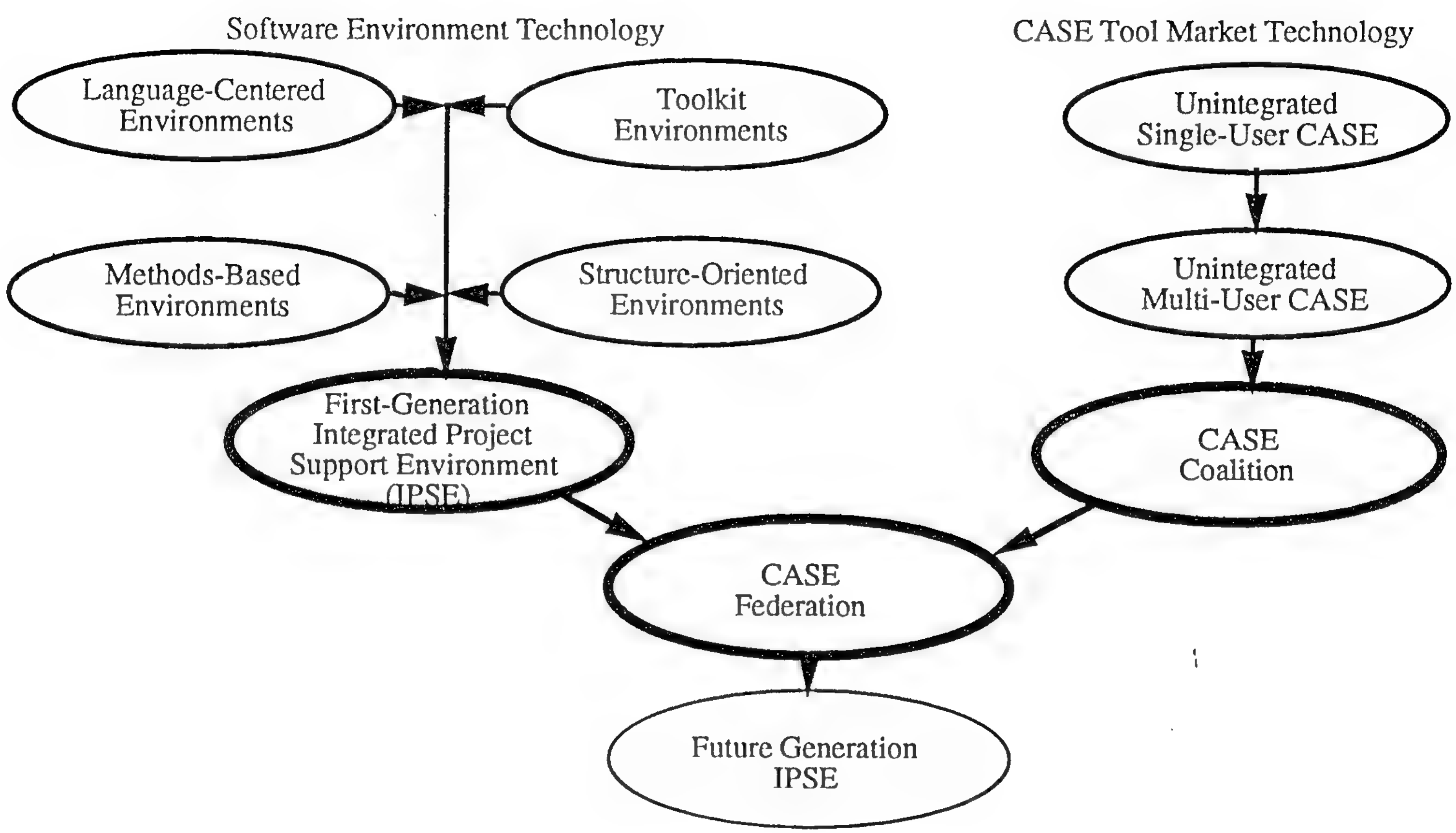
Computing Environment in 1994

Open systems predominate

- Pervasive personal computers
- Technical desktop: Unix workstations
- Business desktop: IBM PC or Macintosh
- Extensive networking
- Client-server model

INTERACTIVE DEVELOPMENT ENVIRONMENTS

CASE Federation Environments likely to emerge



[Feiler - SEI]

Trends

TRW

Architectures:

- Client-Server
- Distribution
- Autonomy
- Interoperability
- Active
- Component-based
- Process-based
- User-tailorable

Processes:

- Architecture-driven
- Reused-based
- Design by Teams
- Cooperative (CSCW)
- Business-driven

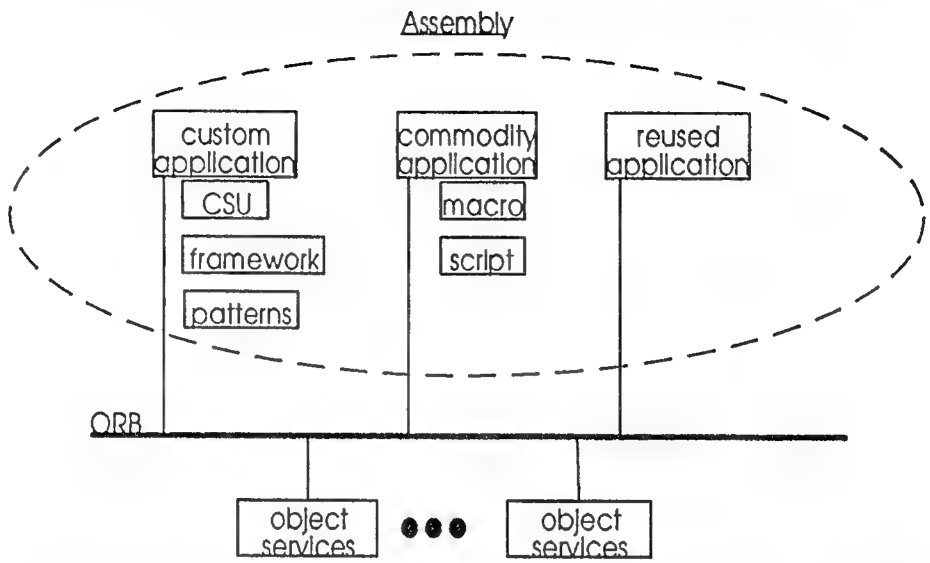
Middleware: The Key to Distributed Megaprogramming

TRW

Developed and Reused Components	Applications and Architecture	
Middleware:	UNAS: Applications Programming Interface and Distributed Runtime Libraries	Distributed, Multiprogram Features
	Ada Predefined Libraries	Single Program Features
Open Systems Standards:	Operating Systems, Networking Protocols and Data Representations	
Execution Platforms	Physical Hardware Resources	

Middleware Software Insulates Applications From The Complexity of Low Level, Systems Programming

CORBA, OLE COM

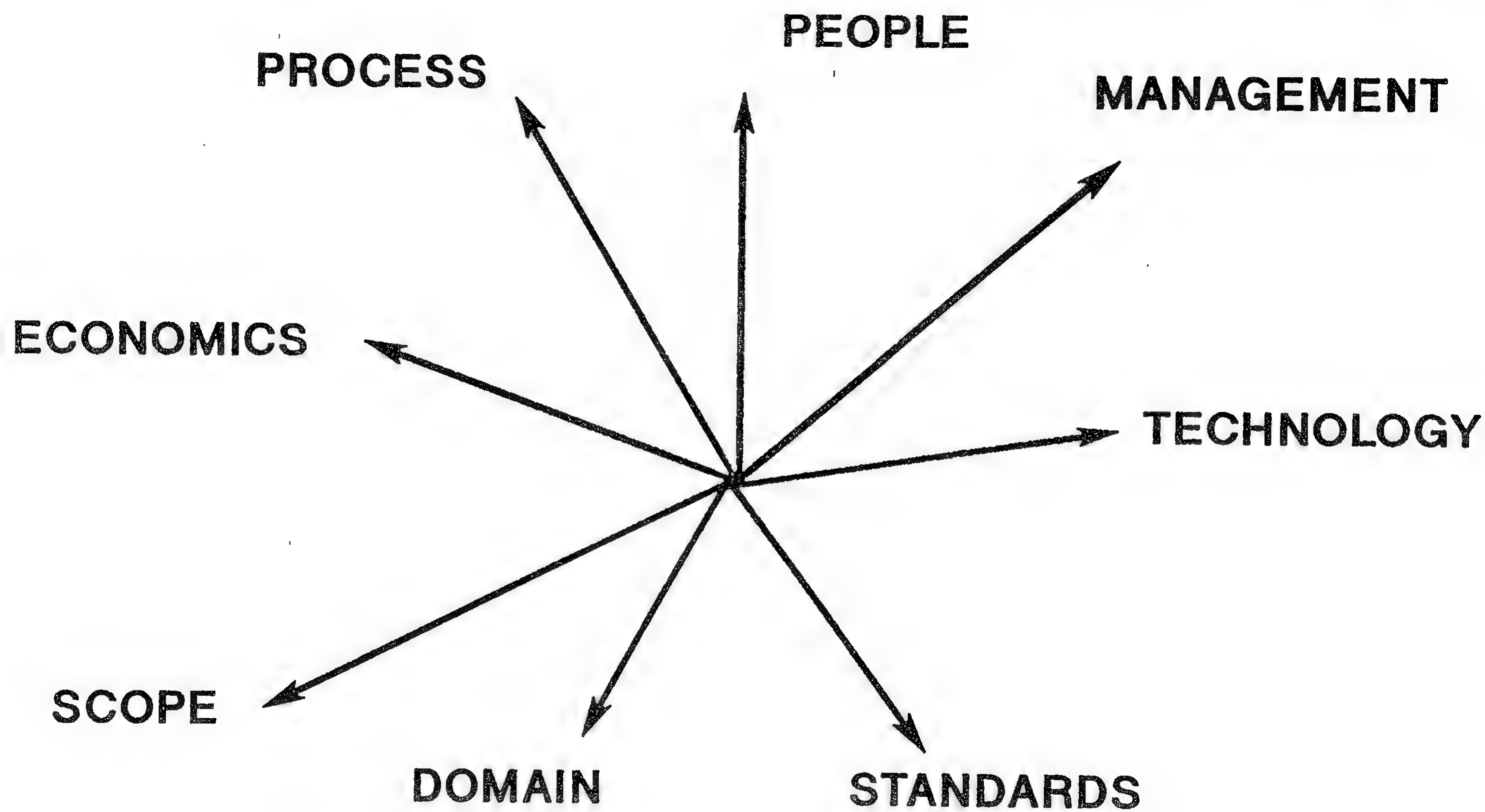


- **Separate programs integration by CORBA, OLE COM**
  - independently authored programs publish object models & operations
  - programs operate each other at coarse or moderate granularity
  - no source code needed
  - no linking or building of (re)used programs
  - commodity products automatable
  - OLE additionally provides a graphics integration metaphor

DoD's Vision of Reuse

- The Vision is to drive the DoD software community from its current "re-invent the software" cycle to a process-driven, domain-specific, architecture-centric, library-based way of constructing software.
- Ingredients:
  - Process-driven reuse - reuse evolves into integral part of Software Engineering process
  - Domain-specific reuse - definition of architectural concepts for each domain plus generic archtiecture (high-level design for a family of related applications).
  - Reusable assets plus assembly methods
  - Providing for tools: program generators, knowledge based approaches, system interconnection tools.
- Key programs: DSSA, STARS

FACTORS INFLUENCING EFFECTIVE REUSE



M. Griss

Software Trends Enabling REUSE

		70's	80's	90's
Software Engineering	Processes	Traditional Waterfall →	Incremental Spiral →	Produce / Manage / Use
	Org's	Programming Teams →		Flexible Learning Organizations
Systems & Software	Methods	ad hoc →	SA / SD → OOA / OOD →	DA / DE
	Environ't	Mainframe & Terminals →	Personal Workstations →	Distributed Computing
	Standards	Portability →	Interoperability →	Reusability
	Appl'n Services	Databases →	4GL; UI Toolkits →	Frameworks
			Spreadsheets →	Problem-Oriented Languages
Languages & Tools	Tools	Construction Tools →	Design Tools →	Generation Tools
	Languages	Procedural (Fortran, C) →	Modular (ADA) → OO (C++) →	Megaprogramming (MIL's, PI)

Systematic Domain Specific Reuse

HEWLETT PACKARD

M. Griss

DOMAIN–SPECIFIC REUSE

Applications will be constructed from domain–specific components within frameworks using builders, generators and glue languages

Business

opportunities

customers

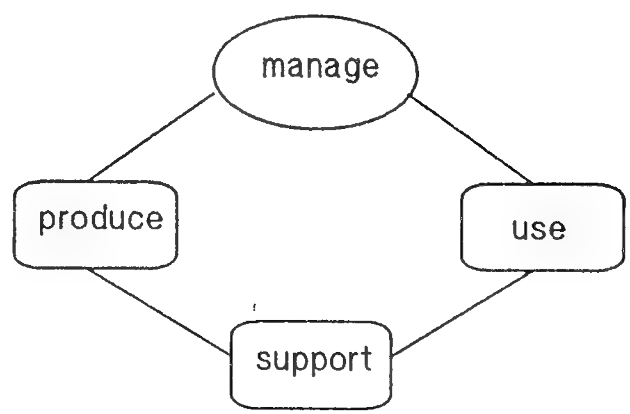
End user programming
Systems integrators
Application developers
Tool builders
Systems programmers

Organization

structures

specialists

lifecycles



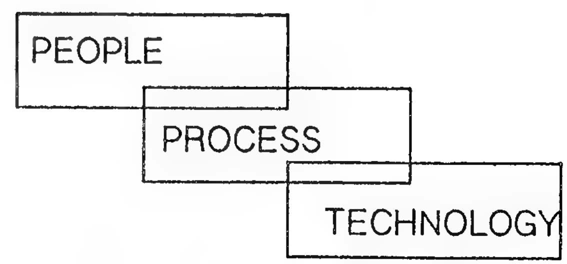
Development Model

methods

tools

libraries

environments



HEWLETT PACKARD

[M. Griss]

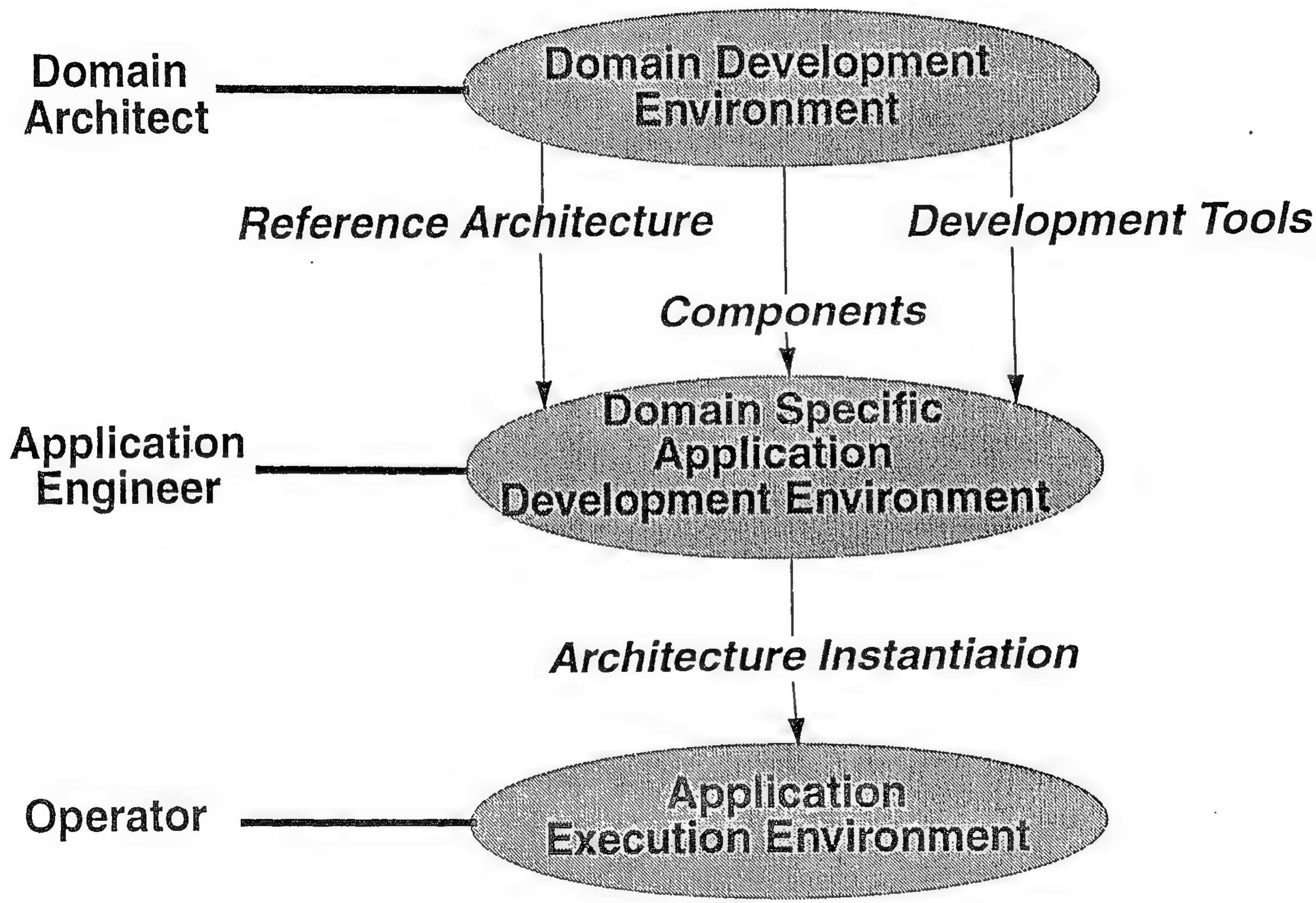
Domain Specific Software Architecture

Software and Intelligent Systems Technology

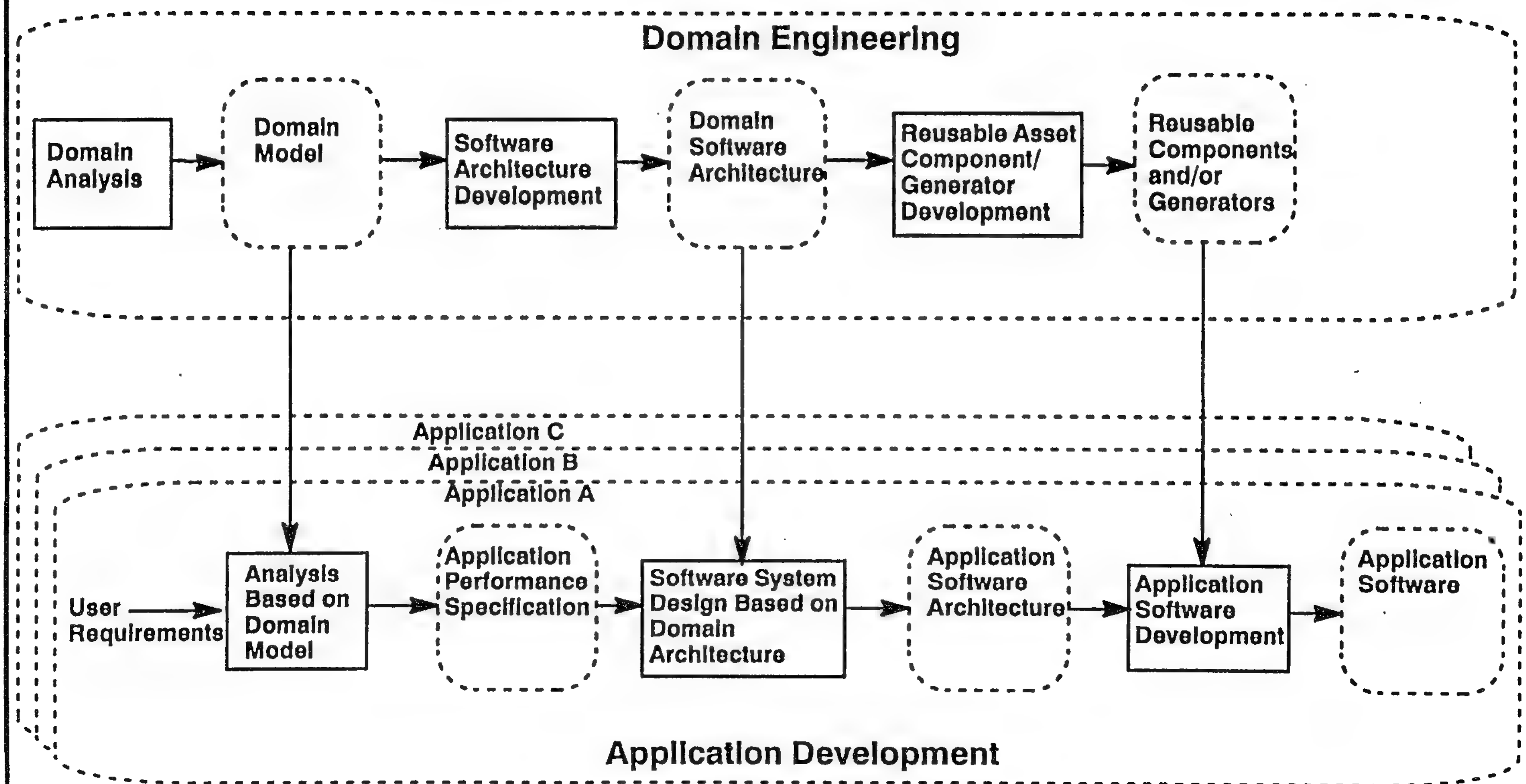
- What is a DSSA?
  - A specification for assemblage of software components,
    - specialized for a particular class of tasks (domain),
    - generalized for effective use across that domain,
    - composed in a standardized structure (topology),
    - effective for building successful applications

- Why do we care?
  - Domain specialization provides needed and valuable constraints:
    - For defining architectures and components
    - For generating or reusing components
    - For accelerating the software development process
    - For technology-based megaprogramming, including domain-specialized development methods and tools.

Three Levels of Systems in DSSA Software Development

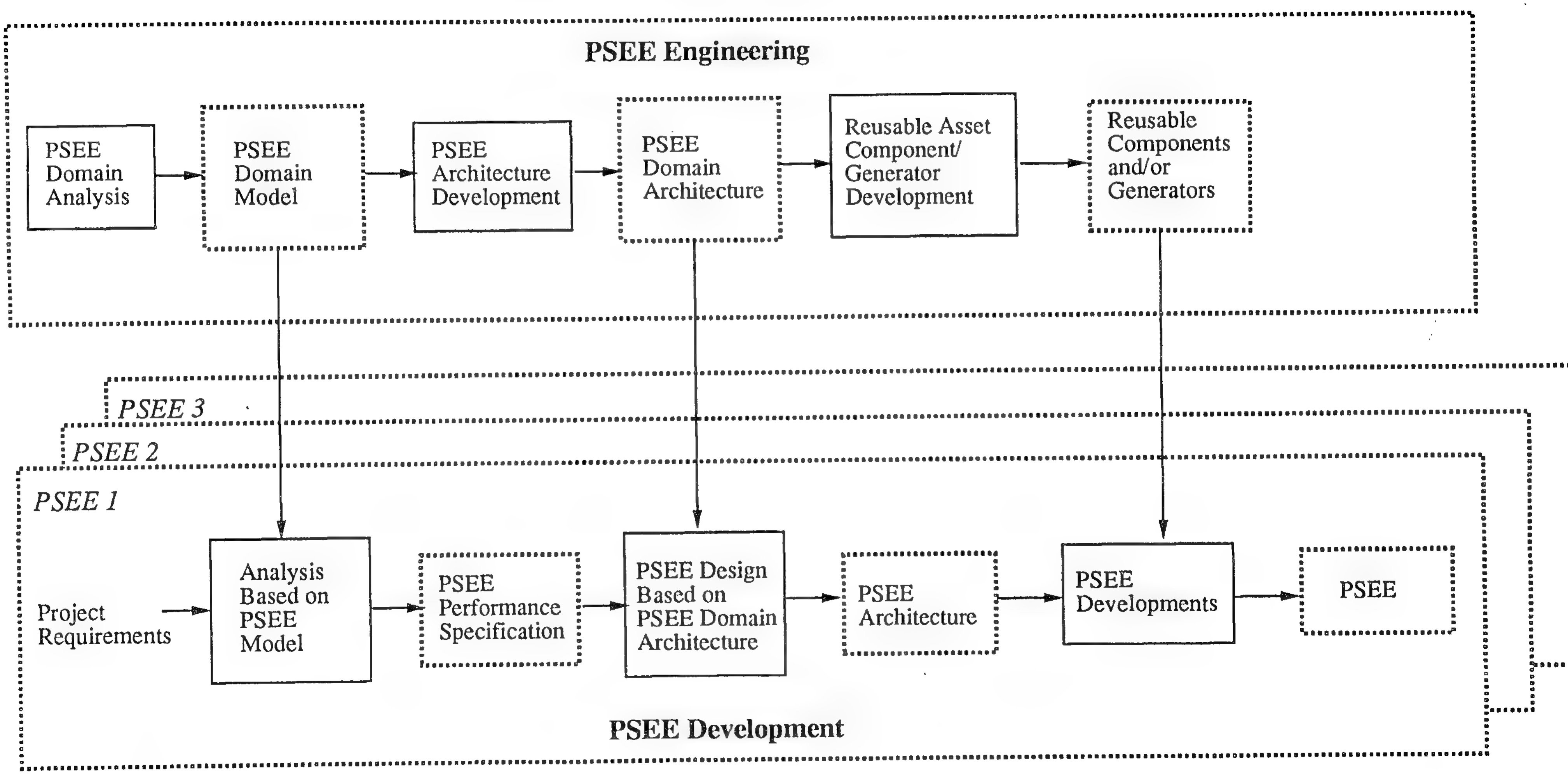


DOMAIN–SPECIFIC REUSE
TWO LIFE-CYCLES



Challenge: Develop technology to support the rapid generation and tailoring of PSEEs.

PSEE Domain Engineering



Conclusions: Megaprogramming

- The pace of megaprogramming is accelerating
  - Future software marketplace reuse-driven
- There are many benefits, but still some risks
  - Need good business case analysis for software product line investments
  - Many fruitful payoff areas (e.g., architectures)
- Exploiting applications domain knowledge is vital
- Need to rethink software processes, specifications, cost models
- Need to rethink software careers, education
- Empowered product line managers are success-critical

The Future of the Software Marketplace

<table>
<tr><td colspan="3">User programming
(100M performers in US)</td></tr>
<tr><td>Application generators
(2M)</td><td>Application composition
(2M)</td><td>System integration
(2M)</td></tr>
<tr><td colspan="3">Infrastructure
(1M)</td></tr>
</table>

[B Boehm]

The future of software careers

User programming			Applications majors
Application generators	Application composition	System integration	Gap
Infrastructure			Computer science majors

OVUM-Market Development Scenario

	1985	Late 1980s	1990	Early 1990s	1995	Late 1990s	2000
Users	Pioneers experimenting	Trials underway. First SDEs appear in large organisations	SDEs used in production by large organisations	Widespread definition of user processes; process groups and central Case task forces in most large/medium organisations	Strong evidence of benefits of integration. Second tier users adopt SDE frameworks		Large organisations have single SDE customised with methods and tools for range of development activities both real-time and IS
Vendors	First Ipse product vendors struggle	Computer systems suppliers enter the market; I-Case suppliers appear	First OMS products delivered	Computer systems suppliers increase presence in market; shakeout of Ipse/I-Case suppliers	Computer systems suppliers compete on strength of SDE support and implementation		Market dominated by major I-Case tool suppliers and computer systems suppliers
PCTE/CIS		PCTE standards efforts gain momentum; CIS launched	First SDEs based on PCTE emerge	PCTE/CIS draw closer together	Framework standards established	Next generation SDE with intelligent process support emerging	
Other new technology directions			Re-use libraries appear; code generation becomes more automated		Systems integration environments appear, pulling together hardware and software engineering tools		

National Information Infrastructure (NII)

TRW

- from article in STSC Crosstalk, July 1994, by K. Alford
- The "electronic superhighway"
- Broadcasted during Pres. Clinton's State of the Union Address on Jan 25, 1994 asking Congress to pass legislation that would assist in the creation of the NII.
- "The National Information Infrastructure: Agenda for Action" is a policy document released by the Clinton administration on Sept 15, 1993.
- Policy document defines the NII as a *"seamless web of communications networks, computers, databases, and consumer electronics that will put vast amounts of information at users' fingertips."*
- What it is **not**:
  - it is not the Internet
  - not yet built
  - is not being built by the federal government
  - has not yet been totally defined
  - will not be free.

National Information Infrastructure (Cont.)

- Examples of NII usage:
  - Doctors diagnosing patients located remotely using (possibly remote) dianostic systems
  - Multi-media training sessions with distributed instructors and students
  - Real-time simulations and mobilization exercises that simultaneously link units and organizations across the nation in a single exercvise.
- No one knows total price; from $100 billion to over $3 trillion
- Government will steer and incentivize but private industry will have to complete the job
- Needed technology:
  - telecommunications technology: broad-band fiber optics, high capacity wireless telephone systems, high speed digital switches, digital compression
  - software (*the largest software project in history)*
  - *information processing*
  - *security and defense issues*
  - *other*

The 2000+ Vision

Probable Characteristics of 2000 SEEs:

- Process Adaptive
  - support for tailoring process assets to organization/project
- Process Supportive
  - fairly complete process definition technology is available
  - significant ability to view process descriptions & monitor process progress during projects
  - full process enactment still emerging
- Reuse-Supportive
  - tailoring to demands of product domains (DSSAs)
- Connections to national repositories are common
- Technology-Supported
- Geographically Distributed Development
- Components acquired from multiple sources, but integrated via Process technology

==> ***Process-Centered SEEs***

Vision of SEEs – Year 2000?

- Allows rapid construction and adaptation by non-expert users
- Supports the full spectrum of project activities and user roles
- User interaction:
  - provides guidance and support for individuals and teams
  - automatically adapts to user expertise
  - interacts with user based on user role in project
  - adapts based on state of project performance
- Takes pro-active participation based on pre-defined instruction
- Supports and validates changes to process
- Interfaces with company and national repositories of assets.

Information System Architecture Evolution

[G. Fox, TRW, 94]

- **Early 1990s - PCs, file-servers, downsizing, sophisticated inexpensive COTS software**
  - Personal/local information augments departmental and corporate information (3 tier architectures)
  - Migration of information from corporate to departmental tiers
  - Tight coupling of system to organizational operation (word processing, EMAIL, budget management, vugraph generation, automated order entry, interactive real time account update, etc.)
  - System architectures begin to parallel organization architectures
- **2000 - Mobile, interconnected communication, large-scale COTS (information managers, EMAIL manipulators)**
  - Sophisticated technical infrastructure
    - Communication/data exchange directly between mobile users
    - Less dependence on tiered architectures
  - Organization cannot function without system
    - Systems never replaced, only evolved

Kaizen - "Everybody improves everything all the time"